COSC 1020: Week 8

Topics: Aggregation

To do: Chapter 8, Lab 8

1

Outline

- **Introduction**
- **Aggregation API**
- **Collections**
- **Algorithm complexity**
- **Further example usage**

2

Outline

- **Introduction**
- **Aggregation API**
- **Collections**
- **Algorithm complexity**
- **Further example usage**

3

Introduction

Motivation

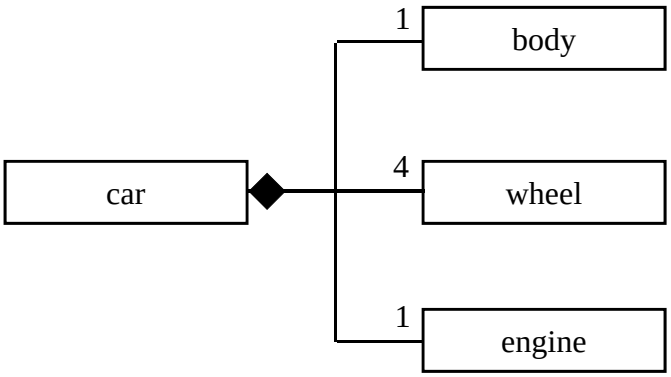
- Our everyday world is full of objects.
  - cars
  - audio/video (A/V) systems
  - computers
- Typically, systems (objects) of even moderate complexity are comprised of subsystems.
  - Car: body, wheels, engine, …
  - A/V system: Receiver, monitor, speakers, …
  - Computers: processor, memory, IO devices, …
- Indeed, the subsystems often are comprised of subsubsystems...

4

Introduction

Motivation

- Example: Because a car is comprised of a body, wheel and engine, we can capture it via UML aggregation (has-a) representation.
- We say that the car aggregates its components.

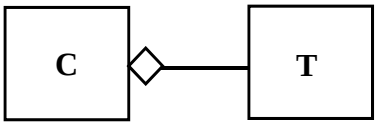


5

Introduction

Aggregation

- A typical software system uses several classes, including the app.
- It is useful to depict the interrelationships that hold.
- Aggregation (has-a): Class C aggregates class T if C has T as an attribute.
- We call C the aggregate class.
- We call T the component or aggregated class.



6

Introduction

Class aggregation

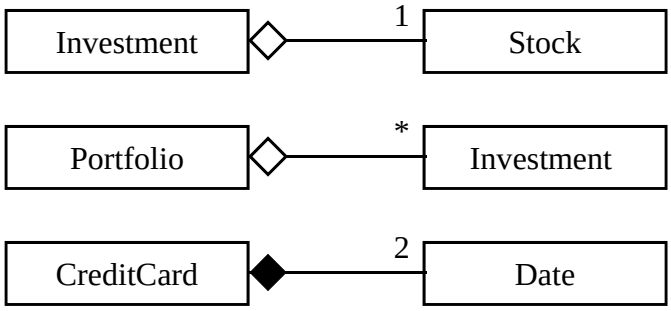
- Our software world is full of objects.
  - Investment
  - Portfolio
  - CreditCard
- Typically, objects of even moderate complexity are comprised of objects.
  - Investment: Stock
  - Portfolio: Investment(s)
  - CreditCard: Dates
- Indeed, the Portfolio is comprised of Investments, which are comprised of Stocks.

7

Introduction

Class aggregation

- Examples: We have introduced three has-a relationships; these can be captured via UML.
- We say that the aggregate class aggregates its component (aggregated) class(es) .

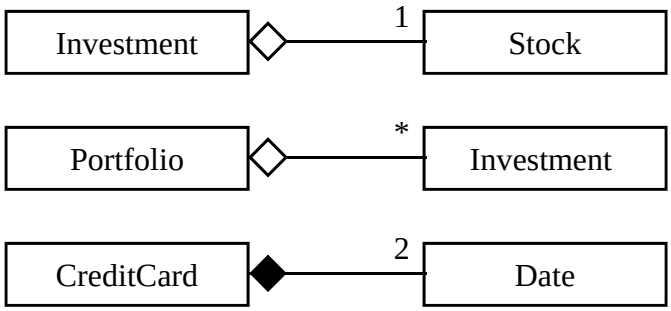


8

Introduction

Multiplicity

- In addition to the aggregate and aggregated classes, aggregation is characterized via multiplicity.
- Multiplicity is the number of attributes in the aggregate class that are of the aggregated type.

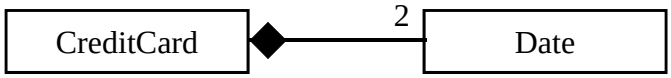


9

Introduction

Composition

- An aggregation between an aggregate class C and an aggregated class T is called a composition if creating an instance of C automatically leads to creating one or more instances of T.

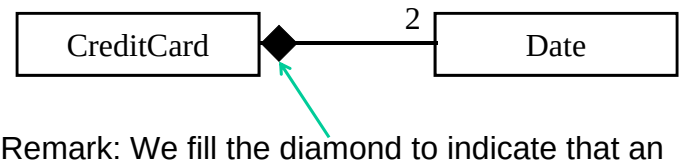


10

Introduction

Composition

- An aggregation between an aggregate class C and an aggregated class T is called a composition if creating an instance of C automatically leads to creating one or more instances of T.



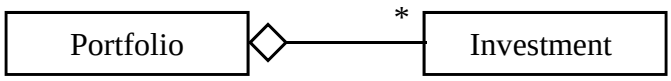
- Remark: We fill the diamond to indicate that an aggregate is a composition.

11

Introduction

Collection

- An aggregation between an aggregate class C and an aggregated class T is called a collection if, rather than forcing all components to be created with the aggregate, an app is allowed to create/add components at any time.



12

Outline

- Introduction
- **Aggregation API**
- Collections
- Algorithm complexity
- Further example usage

13

Aggregation API

Construction of aggregations (non-compositions)

- The aggregate class expects an app to create needed components (aggregated classes)…
- … and then pass the needed references to the aggregate constructor.

14

Aggregation API

Construction of aggregations (non-compositions)

- The aggregate class expects an app to create needed components (aggregated classes)…
- … and then pass the needed references to the aggregate constructor.

Example API

Constructor Summary

Investment(Stock aStock, int aQuantity, double aBValue)
Construct an investment having the passed fields.

15

Aggregation API

Construction of aggregations (non-compositions)

- The aggregate class expects an app to create needed components (aggregated classes)…
- … and then pass the needed references to the aggregate constructor.

Example usage

- Create an investment

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, 24.45);
```

.

Aggregation API

Construction of aggregations (non-compositions)

- The aggregate class expects an app to create needed components (aggregated classes)…
- … and then pass the needed references to the aggregate constructor.

Example usage

- Create an investment

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, 24.45);
```

Alternatively

```
Investment inv=new Investment(new Stock("TD"),15,24.45);
```

.

Aggregation API

Construction of aggregations (non-compositions)

- The aggregate class expects an app to create needed components (aggregated classes)…
- … and then pass the needed references to the aggregate constructor.

Example usage

- Create an investment

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, 24.45);
```

Alternatively

```
Investment inv=new Investment(new Stock("TD"),15,24.45);
```

- Remark: It is permissible to pass a null reference for the component.

18

Aggregation API

Construction of compositions

- In a composition, the aggregate cannot leave it to an app to create components.
- The components are integral to the aggregate and have the same lifetime as the aggregate.
- The constructor creates the components internally as part of its implementation.
- Inspection of the Constructor API does not necessarily reveal presence of aggregated objects.

Aggregation API

Construction of compositions

- In a composition, the aggregate cannot leave it to an app to create components.
- The components are integral to the aggregate and have the same lifetime as the aggregate.
- The constructor creates the components internally as part of its implementation.
- Inspection of the Constructor API does not necessarily reveal presence of aggregated objects.

Example API

Constructor Summary

CreditCard(int no, String aName, double aLimit)

Construct a credit card having the passed...

Aggregation API

Construction of compositions

- In a composition, the aggregate cannot leave it to an app to create components.
- The components are integral to the aggregate and have the same lifetime as the aggregate.
- The constructor creates the components internally as part of its implementation.
- Inspection of the Constructor API does not necessarily reveal presence of aggregated objects.

Example API

Constructor Summary

CreditCard(int no, String aName, double aLimit)

Construct a credit card having the passed...

- Remark: This constructor will implicitly create issue and expiry Date objects (see, java.util.Date).

21

Aggregation API

Construction of compositions

- In a composition, the aggregate cannot leave it to an app to create components.
- The components are integral to the aggregate and have the same lifetime as the aggregate.
- The constructor creates the components internally as part of its implementation.
- Inspection of the Constructor API does not necessarily reveal presence of aggregated objects.

Example usage

- Create a CreditCard

```
CreditCard myCard = new CreditCard(666, “Poe”, 5000.00);
```

22

Aggregation API

Construction of compositions

- In a composition, the aggregate cannot leave it to an app to create components.
- The components are integral to the aggregate and have the same lifetime as the aggregate.
- The constructor creates the components internally as part of its implementation.
- Inspection of the Constructor API does not necessarily reveal presence of aggregated objects.

Example usage

- Create a CreditCard

```
CreditCard myCard = new CreditCard(666, “Poe”, 5000.00);
```

- Remark: Issue and expiry Date objects have been created for myCard.

23

Aggregation API

Component access

- An aggregate class must provide a way to access its components.
- Otherwise, an app would have no way to avail itself to the component objects.

Aggregation API

Component access

- An aggregate class must provide a way to access its components.
- Otherwise, an app would have no way to avail itself to the component objects.
- Example usage

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, stk.getPrice());
Stock stk2 = inv.getStock();
```

Aggregation API

Component access

- An aggregate class must provide a way to access its components.
- Otherwise, an app would have no way to avail itself to the component objects.
- Example usage

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, stk.getPrice());
Stock stk2 = inv.getStock();
CreditCard myCard = new CreditCard(666, "Poe", 5.00);
Date myIssue = myCard.getIssueDate();
Date myExpiry = myCard.getExpiryDate();
```

Aggregation API

Component access

- An aggregate class must provide a way to access its components.
- Otherwise, an app would have no way to avail itself to the component objects.
- Example usage

```
Stock stk = new Stock("TD");
Investment inv = new Investment(stk, 15, stk.getPrice());
Stock stk2 = inv.getStock();
CreditCard myCard = new CreditCard(666, "Poe", 5.00);
Date myIssue = myCard.getIssueDate();
Date myExpiry = myCard.getExpiryDate();
```

- Remark: We do not need to use new, as the components already have been created.

27

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - a reference to the component per se

  or
  - a reference to a copy of the component?

28

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - a reference to the component per se

  or
  - a reference to a copy of the component?
- Either approach will allow the app to retrieve information about the component.

29

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - a reference to the component per se

  or
  - a reference to a copy of the component?
- Either approach will allow the app to retrieve information about the component.
- However,
  - Reference to the component per se: Allows the app to modify (mutate) the original component.
  - Reference to a copy of the component: Does not allow app to modify the component, only the copy.

30

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - Reference to the component: Allows modification of component by app

  or
  - Reference to a copy of component: Does not allow modification of component by app.

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - Reference to the component: Allows modification of component by app

  or
  - Reference to a copy of component: Does not allow modification of component by app.
- Answer (in two parts):

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - Reference to the component: Allows modification of component by app

  or
  - Reference to a copy of component: Does not allow modification of component by app.
- Answer (in two parts):
  - If the app created the component, then it should be able to modify it. →Non-composition aggregations should return reference to component.

Aggregation API

Component access

- A question of interest: Should the component accessor return
  - Reference to the component: Allows modification of component by app

  or
  - Reference to a copy of component: Does not allow modification of component by app.
- Answer (in two parts):
  - If the app created the component, then it should be able to modify it. →Non-composition aggregations should return reference to component.
  - If the app did not create the component, then it should not be able to modify it. → Compositions should return reference to copy of component

34

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to component

35

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to component

```
Investment inv = new Investment(new Stock(“TD”),2,24.45);
output.println(inv.getStock()); // prints TD
Stock stk = inv.getStock();
stk.setSymbol(“RY”);
output.println(stk); // prints Royal
output.println(inv.getStock()); // prints Royal
```

-

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to component

```
Investment inv = new Investment(new Stock("TD"),2,24.45);
output.println(inv.getStock()); // prints TD
Stock stk = inv.getStock();
stk.setSymbol("RY");
output.println(stk); // prints Royal
output.println(inv.getStock()); // prints Royal
```

- Remark: Investment aggregates, but not as a composition.

37

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to copy of component

38

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to copy of component

```
CreditCard myCard = new CreditCard(1, “Wildes”);
output.println(myCard.getIssueDate()); // time is creation time
Date myIssue = myCard.getIssueDate();
myIssue.setTime(0); // see java.util.Date
output.println(myIssue); // time is 0
output.println(myCard.getIssueDate()); // time is creation time
```

39

Aggregation API

Component access and mutation

- Compare the results of an accessor returning reference to component vs. reference to copy of component.
- Reference to copy of component

```
CreditCard myCard = new CreditCard(1, “Wildes”);
output.println(myCard.getIssueDate()); // time is creation time
Date myIssue = myCard.getIssueDate();
myIssue.setTime(0); // see java.util.Date
output.println(myIssue); // time is 0
output.println(myCard.getIssueDate()); // time is creation time
```

- Remark: CreditCard aggregates as a composition.

40

Aggregation API

Aggregation mutation

- Non-composition aggregations typically do not provide mutator methods to change the components.
  - Mutation can be accomplished via the references returned by the collection's accessor methods.

Aggregation API

Aggregation mutation

- Non-composition aggregations typically do not provide mutator methods to change the components.
  - Mutation can be accomplished via the references returned by the collection's accessor methods.
- Compositions must provide mutator methods to change components
  - No reference is provided to the actual components (only reference to a copy of the components).
  - Typically, such mutators come with restrictions (preconditions) in an attempt to maintain consistent state in the component objects.
  - Example: The CreditCard class has a mutator method setExpiryDate that only works if the passed date is not null and is after the issue date.

42

Aggregation API

Recapitulation

Feature in Aggregate	Aggregation (non-composition)	Composition
constructor	Expects a component ref to be passed as a parameter; null okay.	No ref. passed; component created by constructor.
accessor	Must be present in API; returns a component ref.	Must be present in API; returns ref. to component copy.
mutator	Not needed; can mutate via accessor.	Possibly; but with restrictions (preconditions).

43

Outline

- Introduction
- Aggregation API
- **Collections**
- Algorithm complexity
- Further example usage

44

Collections

Basics

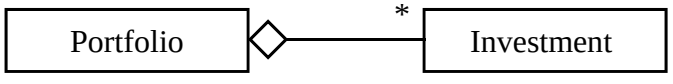
- In many cases, an object has a whole collection of components.
- Moreover, the number of components can vary dynamically.
- Examples
  - A course has a collection of students.
  - A portfolio has a collection of investments.

45

Collections

Basics

- In Java, there are several mechanisms to deal with collections.
  - arrays
  - The Vector class
  - … and others we will see latter.
- Some classes are designed to hide/encapsulate the mechanism that is used to maintain the collection.
  - The Portfolio class.



46

Collections

Construction

- Since the number of components in a collection can vary dynamically...
- … the components can not be created internally by a constructor (as they are with a composition)
- … the components can not be passed as parameters to the constructor (as they are with other aggregates).

47

Collections

Construction

- Since the number of components in a collection can vary dynamically...
- … the components can not be created internally by a constructor (as they are with a composition)
- … the components can not be passed as parameters to the constructor (as they are with other aggregates).
- Standard solution
  - A constructor must be provided that creates an empty collection (and/or one with initial contents).
  - A method must be provided for adding components (to a previously created) collection.

48

Collections

Construction

- Since the number of components in a collection can vary dynamically…
- …should a block of memory sufficient to encompass all anticipated components be allocated upfront at construction?
- …should memory be allocated dynamically as components are added?

49

Collections

Construction

- Since the number of components in a collection can vary dynamically…
- …should a block of memory sufficient to encompass all anticipated components be allocated upfront at construction?
- …should memory be allocated dynamically as components are added?
- In practice, both solutions are encountered
  - Upfront allocation is referred to as static allocation.
  - Allocation as components are added is referred to as dynamic allocation.

50

Collections

Construction

- Example constructors

Portfolio

Constructor Summary

Portfolio(java.lang.String title, int max)

Construct an empty portfolion having the passed name and capable of holding the specified number of investments.

51

Collections

Construction

- Example constructors

Portfolio

Constructor Summary

Portfolio(java.lang.String title, int max)

Construct an empty portfolion having the passed name and capable of holding the specified number of investments.

- Illustrative usage

  Portfolio myPortfolio = new Portfolio(“value”, 10);

52

Collections

Construction

- Example constructors

GlobalCredit

Constructor Summary

GlobalCredit(java.lang.String name)

Constuct a GC processing centre having the name name.

53

Collections

Construction

- Example constructors

GlobalCredit

Constructor Summary

GlobalCredit(java.lang.String name)

Constuct a GC processing centre having the name name.

- Illustrative usage

```
GlobalCredit myCntr = new GlobalCredit("1020Credit");
```

54

Collections

Adding components

- A collection must provide a method for inserting components.
- Often, this method is called “add”.
- When a component is to be added, two conditions for concern might arise.
  1. The collection is full
  2. The component already is present

55

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.
  1. The collection is full
     - Applies only to static allocation. (By definition, dynamic allocation structures are never full).

56

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.
  1. The collection is full
     - Applies only to static allocation. (By definition, dynamic allocation structures are never full).
     - The add method must signal failure to caller.
     - Typically, failure/success signaled by making add have a boolean return.

57

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.
  2. The component is already present
     - Something to be added already was added previously.
     - We distinguish two different contexts.

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.

  2. The component is already present
     - Something to be added already was added previously.
     - We distinguish two different contexts.
       i. List

       ii. Set

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.

  2. The component is already present
     - Something to be added already was added previously.
     - We distinguish two different contexts.
       i. List contexts allow duplication: The component is added to the collection; add method returns void (components always added).

       ii. Set

Collections

Adding components

- When a component is to be added, two conditions for concern might arise.

  2. The component is already present
     - Something to be added already was added previously.
     - We distinguish two different contexts.
       i. List contexts allow duplication: The component is added to the collection; add method returns void (components always added).
       ii. Set contexts do not allow duplication:The duplicate component is not added to the collection; add method returns false.

61

Collections

Adding components

- Method add return summary:
  - Set context (no duplication) + static allocation
    → boolean return
  - Set context (no duplication) + dynamic allocation
    → boolean return
  - List contexts (allows duplication) + static allocation
    → boolean return
  - List contexts (allows duplication) + dynamic allocation
    → void return

62

Collections

Adding components

- Example add methods

Portfolio

Method Summary

boolean add(Investment inv)

Attempt to add the passed investment to this portfolio.

Collections

Adding components

- Example add methods

Portfolio

Method Summary

boolean add(Investment inv)

Attempt to add the passed investment to this portfolio.

- Remarks: Class Portfolio
  - Has a fixed capacity → static allocation.
  - Accepts duplications → list context.

64

Collections

Adding components

- Example add methods

Portfolio

Method Summary

boolean add(Investment inv)

Attempt to add the passed investment to this portfolio.

- Illustrative usage

```
Portfolio myPortfolio = new Portfolio("value", 10);
Investment i1 =new Investment(new Stock("TD"),15,24.45);
boolean success = myPortfolio.add(i1);
```

65

Collections

Adding components

- Example add methods

GlobalCredit

Method Summary

boolean add(CreditCard card)

Attempt to add the passed credit card to this GCC.

66

Collections

Adding components

- Example add methods

GlobalCredit

Method Summary

boolean add(CreditCard card)

Attempt to add the passed credit card to this GCC.

- Remarks: Class GlobalCredit
  - Has no set capacity → dynamic allocation.
  - Does not accept duplicates → set context.

67

Collections

Adding components

- Example add methods

GlobalCredit

Method Summary

boolean add(CreditCard card)

Attempt to add the passed credit card to this GCC.

- Illustrative usage

```
GlobalCredit myCntr = new GlobalCredit("1020Credit");
CreditCard myCard = new CreditCard(1, "Wildes");
boolean success = myCntr.add(myCard);
```

68

Collections

Interlude: Sets vs. Lists

- Sets and lists are two fundamental ADTs (Abstract Data Types) that abstract real world collections of common interest.

69

Collections

Interlude: Sets vs. Lists

- Sets and lists are two fundamental ADTs (Abstract Data Types) that abstract real world collections of common interest.
- Some collections only allow a component to be represented once.
  - The collection of integers [1,10].
  - The collection of credit cards by an issuer with given ID numbers.
  - We refer to such collections as sets.

70

Collections

Interlude: Sets vs. Lists

- Sets and lists are two fundamental ADTs (Abstract Data Types) that abstract real world collections of common interest.
- Some collections only allow a component to be represented once.
  - The collection of integers [1,10].
  - The collection of credit cards by an issuer with given ID numbers.
  - We refer to such collections as sets.
- Some collections allow a component to be represented more than once.
  - The collection of roots of a polynomial.
  - The collection of investments in a stock portfolio.
  - We refer to such collections as lists.

71

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

Create a new portfolio

1. **Acquire necessary info: Prompt user and read response.**
2. **Construct a Portfolio.**

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
```

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
```

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
```

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
```

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
```

loop over number of investments: know num; use for
{ add investment to portfolio

}

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
```

loop over number of investments: know num; use for
{ add investment to portfolio
1. acquire necessary info. to create investment: prompt & read
2. construct investment
3. add investment
}

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{



}
```

80

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{ output.println("Enter stock symbol, number of shares, price per share");
  output.println("press ENTER after each item.");




}
```

81

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{ output.println("Enter stock symbol, number of shares, price per share");
  output.println("press ENTER after each item.");
  pf.add



}
```

82

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print(“Enter number of investments: ”);
int num = input.nextInt( );
output.print(“Enter portfolio name: ”);
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{ output.println(“Enter stock symbol, number of shares, price per share”);
  output.println(“press ENTER after each item.”);
  pf.add
  (
  )
}
```

83

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print(“Enter number of investments: ”);
int num = input.nextInt( );
output.print(“Enter portfolio name: ”);
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{ output.println(“Enter stock symbol, number of shares, price per share”);
  output.println(“press ENTER after each item.”);
  pf.add
  (new Investment(new Stock(input.nextLine()), input.nextInt(), input.nextDouble()
  );
}
```

84

Collections

Adding components

- Example: Read several investments from a user and add them to a portfolio.

```
output.print("Enter number of investments: ");
int num = input.nextInt( );
output.print("Enter portfolio name: ");
String name = input.nextLine( );
Portfolio pf = new Portfolio(name, num);
for (int i = 0; i < num; i++)
{ output.println("Enter stock symbol, number of shares, price per share");
  output.println("press ENTER after each item.");
  pf.add
  (new Investment(new Stock(input.nextLine()), input.nextInt(), input.nextDouble()
  );
}
output.println("Portfolio" + name + "created.");
```

85

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

86

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

Create a new processing centre

1. **Acquire necessary info: Prompt user and read response.**
2. **Construct a GlobalCredit.**

87

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
```

88

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
```

89

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
```

loop over set of cards: Don't know cardinality in advance;
but can define sentinel → use while (true) + break
(add card to processing centre

)

90

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
```

loop over set of cards: Don't know cardinality in advance; but can define sentinel → use while (true) + break
(add card to processing centre
1. Acquire necessary info. To create card: prompt & read
2. Break if card num == 0, the sentinel
3. Construct card
4. Add card
)

91

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{




}
```

92

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{ output.print("Enter card number (0 to quit): ");
  int num = input.nextInt( );




}
```

93

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{ output.print("Enter card number (0 to quit): ");
  int num = input.nextInt( );
  if (num == 0) break; // this is the way out of loop




}
```

94

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{ output.print("Enter card number (0 to quit): ");
  int num = input.nextInt( );
  if (num == 0) break; // this is the way out of loop
  output.print("Enter name: ");
  String who = input.nextLine( );

}
```

95

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{ output.print("Enter card number (0 to quit): ");
  int num = input.nextInt( );
  if (num == 0) break; // this is the way out of loop
  output.print("Enter name: ");
  String who = input.nextLine( );
  gcc.add(new CreditCard(num, who));
}
```

96

Collections

Adding components

- Example: Read several credit cards from user and add them to a processing centre.

```
output.print("Enter name of processing centre: ");
String name = input.nextLine( );
GlobalCredit gcc  = new GlobalCredit(name);
while (true)
{ output.print("Enter card number (0 to quit): ");
  int num = input.nextInt( );
  if (num == 0) break; // this is the way out of loop
  output.print("Enter name: ");
  String who = input.nextLine( );
  gcc.add(new CreditCard(num, who));
}
output.println("Processing centre " + name + "created.");
```

97

Collections

Component access

- A collection must provide a method for accessing its components.
- For the simplest aggregations and compositions, e.g.,
  - Investment
  - CreditCard
- The API documented one accessor per component, with the components distinguished by name, e.g.,
  - getStock
  - getIssueDate

98

Collections

Component access

- A collection must provide a method for accessing its components.
- For the simplest aggregations and compositions, e.g.,
  - Investment
  - CreditCard
- The API documented one accessor per component, with the components distinguished by name, e.g.,
  - getStock
  - getIssueDate
- The approach seen so far will not work for collections, e.g.,
  - Portfolio holds numerous un-named investments.

99

Collections

Component access

- We consider component access for collections via systematic traversal in two ways:
  1. Indexed traversal:
  2. Iterator traversal:
- The API of a collection should document methods for one or both approaches.

100

Collections

Component access

- We consider component access for collections via systematic traversal in two ways:
  1. Indexed traversal: Think of the components as numbered, i.e., having a numerical index; to access, ask for components by index.
  2. Iterator traversal:
- The API of a collection should document methods for one or both approaches.

101

Collections

Component access

- We consider component access for collections via systematic traversal in two ways:
  1. Indexed traversal: Think of the components as numbered, i.e., having a numerical index; to access, ask for components by index.
  2. Iterator traversal: The traversal operates at a higher level of abstraction than index-based. It automatically guarantees that all elements are visited without missing one and without repeated visits to the same element. The client is not concerned with indexing or ordering of elements
- The API of a collection should document methods for one or both approaches.

102

Collections

Component access: Indexed traversal

- Think of the components as having a numerical index
- It is not necessary the ordering of the indices reflect the order of insertion into the collection.
- What is necessary is that there be a 1 to 1 mapping between indices and components.
- Remark: Recall our earlier discussion of arrays.

103

Collections

Component access: Indexed traversal

- Two methods are of interest.

104

Collections

Component access: Indexed traversal

- Two methods are of interest.
  1. int size( )

     Returns the number of components in the collection, with 0 return indicating empty. For non-empty collections, the legal range of indices is [0,size-1].

105

Collections

Component access: Indexed traversal

- Two methods are of interest.
  1. int size( )

     Returns the number of components in the collection, with 0 return indicating empty. For non-empty collections, the legal range of indices is [0,size-1].
  2. type get(int index)

     Returns a reference to the component with the passed index. The return has the type of the component.

106

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

107

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
```

108

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (int j=0; j<pf.size(); j++)
{



}
```

109

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (int j=0; j<pf.size(); j++)
{   output.print(pf.get(j).getStock().getSymbol() +"\t");



}
```

110

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (int j=0; j<pf.size(); j++)
{   output.print(pf.get(j).getStock().getSymbol() +"\t");
    output.print(pf.get(j).getQty() + "\t");

}
```

111

Collections

Component access: Indexed traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (int j=0; j<pf.size(); j++)
{   output.print(pf.get(j).getStock().getSymbol() +"\t");
    output.print(pf.get(j).getQty() + "\t");
    output.println(pf.get(j).getBookValue());
}
```

112

Collections

Component access: Iterator-based traversal

- An *enhanced* for loop is employed.
- Let E be the type of element e that are components of a collection bag

```
for (E e : bag)
{
        // visit element e
}
```

- Remark: The colon, **:**, in the syntax can be read as “in”.

Collections

Component access: Iterator-based traversal

- An *enhanced* for loop is employed.
- Let E be the type of element e that are components of a collection bag

```
for (E e : bag)
{
        // visit element e
}
```

- Remark: The colon, **:**, in the syntax can be read as “in”.
- This type of traversal is simple, but inflexible: You must traverse; you can’t ask about a particular element.
- To tell if a class supports iterator-based traversal, consult the API and see if it implements the Iterable interface.

114

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

115

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
```

116

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (Investment inv : pf)
{



}
```

117

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (Investment inv : pf)
{   output.print(inv.getStock().getSymbol() +"\t");



}
```

118

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (Investment inv : pf)
{   output.print(inv.getStock().getSymbol() +"\t");
    output.print(inv.getQty() + "\t");

}
```

119

Collections

Component access: Iterator-based traversal

- Example usage: Iterate over the components of a Portfolio and report on contents.

```
Portfolio pf = Portfolio.getRandom();
for (Investment inv : pf)
{   output.print(inv.getStock().getSymbol() +"\t");
    output.print(inv.getQty() + "\t");
    output.println(inv.getBookValue());
}
```

120

Collections

Component search

- Search is concerned with determining whether or not a given collection contains a specific, given component.
- Examples
  - Does a Portfolio contain a Stock with a given symbol?
  - Does a GlobalCredit contain a CreditCard with a given num?

Collections

Component search

- Search is concerned with determining whether or not a given collection contains a specific, given component.
- Examples
  - Does a Portfolio contain a Stock with a given symbol?
  - Does a GlobalCredit contain a CreditCard with a given num?
- There are two possible outcomes of a search.
  1. Success: The desired component is found and is returned as a reference.
  2. Failure: The desired component is not found, null may be returned.

Collections

Component search

- Search is concerned with determining whether or not a given collection contains a specific, given component.
- Examples
  - Does a Portfolio contain a Stock with a given symbol?
  - Does a GlobalCredit contain a CreditCard with a given num?
- There are two possible outcomes of a search.
  1. Success: The desired component is found and is returned as a reference.
  2. Failure: The desired component is not found, null may be returned.
- Remark: There may be more than one component in a collection that satisfies the search conditions.

123

Collections

Component search

- We can implement search by leveraging either of the access methods that we have found for collections (indexed or iterator).

124

Collections

Component search

- We can implement search by leveraging either of the access methods that we have found for collections (indexed or iterator).
  - Set up a loop that visits every component of the collection.

125

Collections

Component search

- We can implement search by leveraging either of the access methods that we have found for collections (indexed or iterator).
  - Set up a loop that visits every component of the collection.
  - For each component, check to see if it satisfies the search condition.

126

Collections

Component search

- We can implement search by leveraging either of the access methods that we have found for collections (indexed or iterator).
  - Set up a loop that visits every component of the collection.
  - For each component, check to see if it satisfies the search condition.
  - If so, then exit the loop and return the component.

127

Collections

Component search

- We can implement search by leveraging either of the access methods that we have found for collections (indexed or iterator).
  - Set up a loop that visits every component of the collection.
  - For each component, check to see if it satisfies the search condition.
  - If so, then exit the loop and return the component.
  - If the end of the collection is encountered, then exit the loop and return null.

128

Collections

Component search

- Example (partial) implementation

129

Collections

Component search

- Example (partial) implementation

```
// assume inv a valid investment and pf a valid portfolio
```

130

Collections

Component search

- Example (partial) implementation

```
// assume inv a valid investment and pf a valid portfolio
boolean found = false;
```

131

Collections

Component search

- Example (partial) implementation

```
// assume inv a valid investment and pf a valid portfolio
boolean found = false;
for (int j=0; j<pf.size(); j++)
{


}
```

132

Collections

Component search

- Example (partial) implementation

```
// assume inv a valid investment and pf a valid portfolio
boolean found = false;
for (int j=0; j<pf.size(); j++)
{       Investment current = pf.get(j);


}
```

133

Collections

Component search

- Example (partial) implementation

```
// assume inv a valid investment and pf a valid portfolio
boolean found = false;
for (int j=0; j<pf.size(); j++)
{       Investment current = pf.get(j);
        if (current.equals(inv))
          found = true;
}
```

134

Outline

- Introduction
- Aggregation API
- Collections
- **Algorithm complexity**
- Further example usage

135

Algorithm complexity

Analysis of algorithms

- In general, execution time increases with the size of input.
- For example, searching for an item in a collection takes longer as the size of the collection increases.

136

Algorithm complexity

Analysis of algorithms

- In general, execution time increases with the size of input.
- For example, searching for an item in a collection takes longer as the size of the collection increases.
- To formalize
  - Let n be the size of the input (e.g., collection size).
  - Let $T(n)$ represent the running time of an algorithm as a function of input size, n.

137

Algorithm complexity

Example

- How long will it take our simple search algorithm to execute?

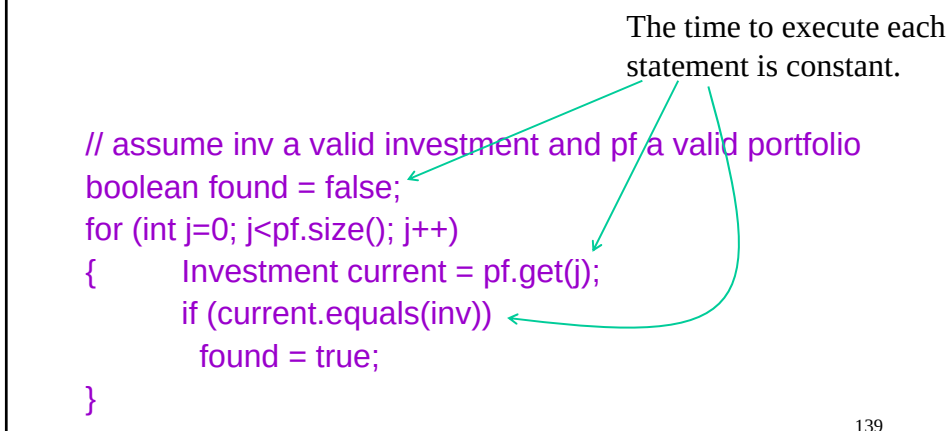
```
// assume inv a valid investment and pf a valid portfolio
boolean found = false;
for (int j=0; j<pf.size(); j++)
{       Investment current = pf.get(j);
        if (current.equals(inv))
          found = true;
}
```

138

Algorithm complexity

Example

- How long will it take our simple search algorithm to execute?

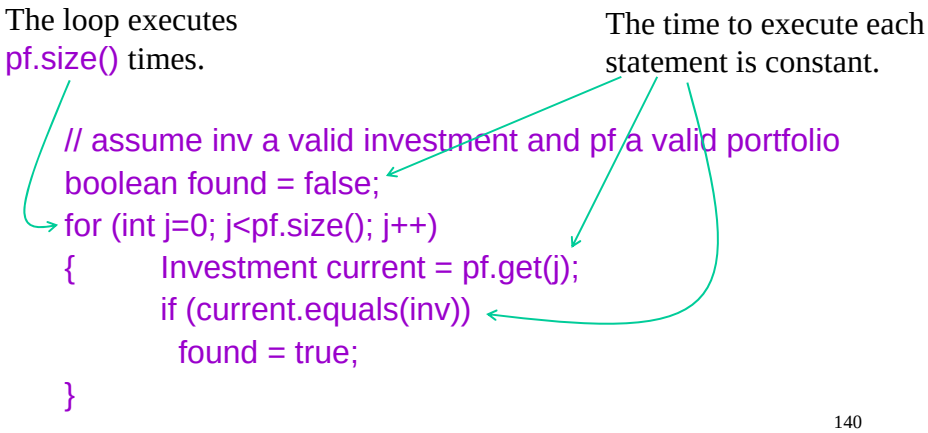


139

Algorithm complexity

Example

- How long will it take our simple search algorithm to execute?

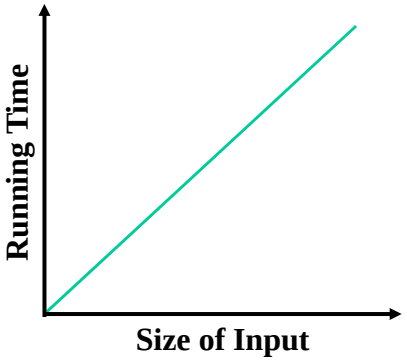


140

Algorithm complexity

Example

- Overall, we have shown that run time, $T(n)$, essentially depends directly on n = pf.size().
- In particular, the running time of our search algorithm depends linearly on the size of input n.

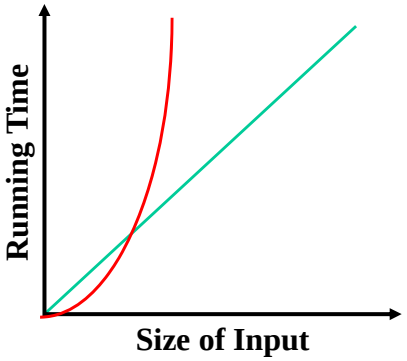


141

Algorithm complexity

Alternatively

- Other algorithms might execute at a rate that grows more rapidly than linearly with size of input.
- Such an algorithm is superlinear in complexity.
- In general, linear is preferred over superlinear.

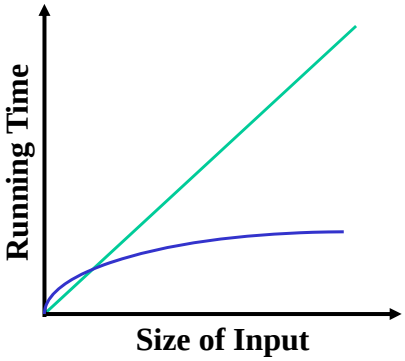


142

Algorithm complexity

Alternatively

- Yet other algorithms might have execution rates that grow slower than linearly with size of input.
- Such an algorithm is sublinear in complexity.
- In general, sublinear is preferable over linear.

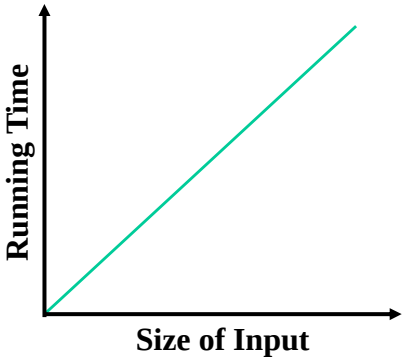


143

Algorithm complexity

Back to our search algorithm

- Overall, we have shown that $T(n)$ essentially depends directly on n.
- In particular, the running time of our search algorithm depends linearly on the size of input n.
- We *write* $T(n)$ is $O(n)$.
- We *say* $T(n)$ is Big-Oh n.

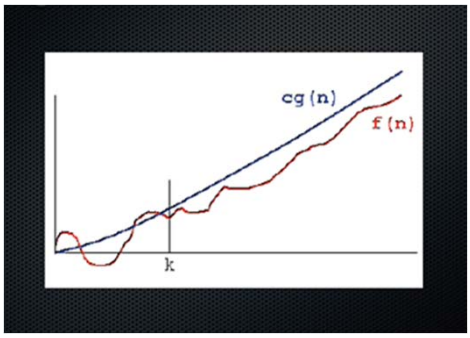


144

Algorithm complexity

Big-Oh

- Informally, *f(n)* is *O(g(n))* if *f(n)* is some constant times *g(n)*.

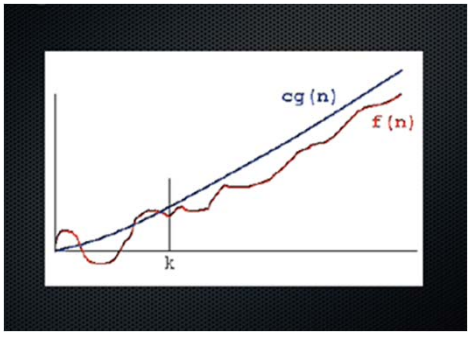


145

Algorithm complexity

Big-Oh

- Informally, *f(n)* is *O(g(n))* if *f(n)* is some constant times *g(n)*.
- Formally, *f(n)* is *O(g(n))* iff there exist positive constants C and K such that
  $$f(n) \le C \cdot g(n)$$
  for all $n \ge K$.

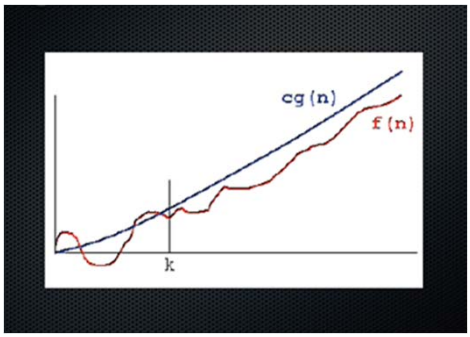


146

Algorithm complexity

Big-Oh

- Informally, $f(n)$ is $O(g(n))$ if $f(n)$ is some constant times $g(n)$.
- Formally, $f(n)$ is $O(g(n))$ iff there exist positive constants C and K such that
  $$f(n) \leq C \cdot g(n)$$
  for all $n \geq K$.
- Big-Oh gives us an estimate of how fast running time grows as n grows and is thus a useful and standard characterization of algorithm efficiency.



147

Algorithm complexity

Exhaustive search

- The search algorithm we have considered is called *exhaustive search*.
- We consider each component of the collection, start to finish, until one matches the target or we reach the end.

148

Algorithm complexity

Exhaustive search

- The search algorithm we have considered is called *exhaustive search*.
- We consider each component of the collection, start to finish, until one matches the target or we reach the end.
- We say that such a search algorithm is linear in the number of components in the collection.
  - For a collection of N components, we must consider N items in the worst case.

Algorithm complexity

Exhaustive search

- The search algorithm we have considered is called *exhaustive search*.
- We consider each component of the collection, start to finish, until one matches the target or we reach the end.
- We say that such a search algorithm is linear in the number of components in the collection.
  - For a collection of N components, we must consider N items in the worst case.
- Interestingly, better (sublinear) search algorithms are available, although they rely on some structuring of the collection.
  - You will hear more about such improvements as your computer science education advances.

150

Outline

- Introduction
- Aggregation API
- Collections
- Algorithm complexity
- **Further example usage**

151

Further example usage

Using Investment

152

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  {
```

153

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
```

154

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
```

155

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
```

156

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
```

157

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
    output.println("inv1.getQty()= " + inv1.getQty()); // 100
```

158

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
    output.println("inv1.getQty()= " + inv1.getQty()); // 100
    output.println("inv1.getBookValue()= " + inv1.getBookValue()); // TD$
```

159

Further example usage

```
Using Investment
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
    output.println("inv1.getQty()= " + inv1.getQty()); // 100
    output.println("inv1.getBookValue()= " + inv1.getBookValue()); // TD$
    output.println("inv1.getStock().toString()= " +
                   inv1.getStock().toString()); // TD…
```

160

Further example usage

Using Investment

```
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
    output.println("inv1.getQty()= " + inv1.getQty()); // 100
    output.println("inv1.getBookValue()= " + inv1.getBookValue()); // TD$
    output.println("inv1.getStock().toString()= " +
                   inv1.getStock().toString()); // TD…
    output.println("inv1.getStock().getSymbol()= " +
                   inv1.getStock().getSymbol()); // TD
```

161

Further example usage

Using Investment

```
// assume the usual
import type.lib.*;
public class InvestmentTest
{ public static void main(String[ ] args)
  { Stock stk1 = new Stock("TD");
    output.println("stk1= " + stk1.toString()); // TD…
    Investment inv1 = new Investment(stk1,100,stk1.getPrice());
    output.println("inv1= " + inv1.toString()); // TD inv…
    output.println("inv1.getQty()= " + inv1.getQty()); // 100
    output.println("inv1.getBookValue()= " + inv1.getBookValue()); // TD$
    output.println("inv1.getStock().toString()= " +
                   inv1.getStock().toString()); // TD…
    output.println("inv1.getStock().getSymbol()= " +
                   inv1.getStock().getSymbol()); // TD
   // continued on next slide
```

162

Further example usage

```
Using Investment
    // continued from previous slide
```


Further example usage

```
Using Investment
    // continued from previous slide
    Stock stk2 = new Stock("BMO");
    output.println("stk2= " + stk2.toString()); // BMO…
```

Further example usage

Using Investment

```
// continued from previous slide
Stock stk2 = new Stock("BMO");
output.println("stk2= " + stk2.toString()); // BMO…
Investment inv2 = new Investment(stk2,100,stk2.getPrice());
output.println("inv2= " + inv2.toString()); // BMO inv…
```

165

Further example usage

Using Investment

```
// continued from previous slide
Stock stk2 = new Stock("BMO");
output.println("stk2= " + stk2.toString()); // BMO…
Investment inv2 = new Investment(stk2,100,stk2.getPrice());
output.println("inv2= " + inv2.toString()); // BMO inv…
output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
```

166

Further example usage

Using Investment

```
// continued from previous slide
Stock stk2 = new Stock("BMO");
output.println("stk2= " + stk2.toString()); // BMO…
Investment inv2 = new Investment(stk2,100,stk2.getPrice());
output.println("inv2= " + inv2.toString()); // BMO inv…
output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
Stock stk3 = inv1.getStock();
output.println("stk3= " + stk3.toString()); // TD…
```

167

Further example usage

Using Investment

```
// continued from previous slide
Stock stk2 = new Stock("BMO");
output.println("stk2= " + stk2.toString()); // BMO…
Investment inv2 = new Investment(stk2,100,stk2.getPrice());
output.println("inv2= " + inv2.toString()); // BMO inv…
output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
Stock stk3 = inv1.getStock();
output.println("stk3= " + stk3.toString()); // TD…
stk3.setSymbol("RY");
output.println("reset stk3's symbol to RY");
output.println("stk3= " + stk3.toString()); // RY…
```

168

Further example usage

```
Using Investment
    // continued from previous slide
    Stock stk2 = new Stock("BMO");
    output.println("stk2= " + stk2.toString()); // BMO…
    Investment inv2 = new Investment(stk2,100,stk2.getPrice());
    output.println("inv2= " + inv2.toString()); // BMO inv…
    output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
   Stock stk3 = inv1.getStock();
   output.println("stk3= " + stk3.toString()); // TD…
   stk3.setSymbol("RY");
   output.println("reset stk3's symbol to RY");
   output.println("stk3= " + stk3.toString()); // RY…
   output.println("inv1.getStock()= " + inv1.getStock().toString())
```

169

Further example usage

```
Using Investment
    // continued from previous slide
    Stock stk2 = new Stock("BMO");
    output.println("stk2= " + stk2.toString()); // BMO…
    Investment inv2 = new Investment(stk2,100,stk2.getPrice());
    output.println("inv2= " + inv2.toString()); // BMO inv…
    output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
   Stock stk3 = inv1.getStock();
   output.println("stk3= " + stk3.toString()); // TD…
   stk3.setSymbol("RY");
   output.println("reset stk3's symbol to RY");
   output.println("stk3= " + stk3.toString()); // RY…
   output.println("inv1.getStock()= " + inv1.getStock().toString());//RY…
```

170

Further example usage

```
Using Investment
    // continued from previous slide
    Stock stk2 = new Stock("BMO");
    output.println("stk2= " + stk2.toString()); // BMO…
    Investment inv2 = new Investment(stk2,100,stk2.getPrice());
    output.println("inv2= " + inv2.toString()); // BMO inv…
    output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
   Stock stk3 = inv1.getStock();
   output.println("stk3= " + stk3.toString()); // TD…
   stk3.setSymbol("RY");
   output.println("reset stk3's symbol to RY");
   output.println("stk3= " + stk3.toString()); // RY…
   output.println("inv1.getStock()= " + inv1.getStock().toString());//RY…
   // continued on next slide
```

171

Further example usage

```
Using Investment
   // continued from previous slide




  }
}
```

172

Further example usage

```
Using Investment
   // continued from previous slide
   stk2.setSymbol("RY");
   output.println("reset stk2's symbol to RY");
   output.println("stk2= " + stk2.toString()); // RY…


  }
 }
```


Further example usage

```
Using Investment
   // continued from previous slide
   stk2.setSymbol("RY");
   output.println("reset stk2's symbol to RY");
   output.println("stk2= " + stk2.toString()); // RY…
   output.println("inv2.getStock()=" + inv2.getStock().toString());

  }
}
```

Further example usage

```
Using Investment
   // continued from previous slide
   stk2.setSymbol("RY");
   output.println("reset stk2's symbol to RY");
   output.println("stk2= " + stk2.toString()); // RY…
   output.println("inv2.getStock()=" + inv2.getStock().toString()); //RY…

  }
}
```


Further example usage

```
Using Investment
   // continued from previous slide
   stk2.setSymbol("RY");
   output.println("reset stk2's symbol to RY");
   output.println("stk2= " + stk2.toString()); // RY…
   output.println("inv2.getStock()=" + inv2.getStock().toString()); //RY…
   output.println("inv1.equals(inv2)= " + inv1.equals(inv2)); // false
  }
}
```

Further example usage

Using Investment: Lessons learned

- We have seen that the Investment constructor uses the Stock object passed as a component of the Investment.
- We also have seen that the getStock accessor method returns a reference to this component.
- This allows the user to change the state of the Investment object, sometimes with strange results.
- Investment could protect against such changes by
  - Setting its component to a copy of the passed Stock object in the constructor
  - Returning a copy of its component in getStock

177

Further example usage

Using Investment: Lessons learned

- In general, to ensure that users cannot change components of an object without the object's methods,…
- …one must make a deep copy of the components passed to/from the object.
- A deep copy is a copy where the subcomponents and subsubcomponents also are copied.

178

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  {
```

179

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
     output.println(ptf1.toString()); // My e.g. Portfolio: 0
```

180

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    output.println(ptf1.toString()); // My e.g. Portfolio: 0
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

181

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    output.println(ptf1.toString()); // My e.g. Portfolio: 0
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
```

182

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    output.println(ptf1.toString()); // My e.g. Portfolio: 0
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

183

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    output.println(ptf1.toString()); // My e.g. Portfolio: 0
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println(ptf1.toString()); // My e.g. Portfolio: 3
```

184

Further example usage

Iterating through a Portfolio

```
// assume the usual template stuff
import type.lib.*;
public class PortfolioTest
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    output.println(ptf1.toString()); // My e.g. Portfolio: 0
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println(ptf1.toString()); // My e.g. Portfolio: 3
   // continued on next slide
```

185

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
```

186

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
```

187

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    {

    }
```

188

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
```

189

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
```

190

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
    for (Investment inv : ptf1)
```

191

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
    for (Investment inv : ptf1)
      output.println(inv);
```

192

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
    for (Investment inv : ptf1)
      output.println(inv);
    stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,150,stk1.getPrice()));
```

193

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
    for (Investment inv : ptf1)
      output.println(inv);
    stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,150,stk1.getPrice()));
    output.println("Traverse again over investments using iterator.);
```

194

```
} }
```

Further example usage

Iterating through a Portfolio

```
    // continued from previous slide
    output.println("Traverse over investments using indexing");
    int ptf1cnt = ptf1.size();
    for (int j=0; j<ptf1cnt; j++)
    { Investment inv = ptf1.get(j);
      output.println(inv);
    }
    output.println("Traverse over investments using iterator.");
    for (Investment inv : ptf1)
      output.println(inv);
    stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,150,stk1.getPrice()));
    output.println("Traverse again over investments using iterator.");
    for (Investment inv : ptf1)
      output.println(inv);
} }
```

195

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
```

196

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Traverse over investments using indexing
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse again over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=59.99
TD Toronto-Dominion Bank QTY=150 BV=81.32
%
```

197

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Traverse over investments using indexing
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse again over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=59.99
TD Toronto-Dominion Bank QTY=150 BV=81.32
%
```

198

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Traverse over investments using indexing
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse again over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=59.99
TD Toronto-Dominion Bank QTY=150 BV=81.32
%
```

199

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Traverse over investments using indexing
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse again over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=59.99
TD Toronto-Dominion Bank QTY=150 BV=81.32
%
```

200

Further example usage

Iterating through a Portfolio: Test

```
% java PortfolioTest
My e.g. Portfolio: 0
My e.g. Portfolio: 3
Traverse over investments using indexing
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
Traverse again over investments using iterator
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=39.03
RY Royal Bank of Canada QTY=100 BV=59.99
TD Toronto-Dominion Bank QTY=150 BV=81.32
%
```

201

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
 {
```

202

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio(“My e.g. Portfolio”, 10);
```

203

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio(“My e.g. Portfolio”, 10);
    Stock stk1 = new Stock(“TD”);
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

204

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
```

205

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

206

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println(ptf1.toString()); // My e.g. Portfolio: 3
```

207

Further example usage

A Portfolio can be modified in strange ways

```
// assume the usual
import type.lib.*;
public class PortfolioTest2
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println(ptf1.toString()); // My e.g. Portfolio: 3
   // continued on next slide
```

208

Further example usage

A Portfolio can be modified in strange ways

```
   // continued from previous slide









 }
}
```

209

Further example usage

A Portfolio can be modified in strange ways

```
   // continued from previous slide
   for (Investment inv : ptf1)
      output.println(inv);
   output.println();









 }
}
```

210

Further example usage

A Portfolio can be modified in strange ways

```
   // continued from previous slide
   for (Investment inv : ptf1)
      output.println(inv);
   output.println();
   Investment inv1 = ptf1.get(0);
```

```
 }
}
```

211

Further example usage

A Portfolio can be modified in strange ways

```
   // continued from previous slide
   for (Investment inv : ptf1)
      output.println(inv);
   output.println();
   Investment inv1 = ptf1.get(0);
   stk1 = inv1.getStock();
   stk1.setSymbol("RY");
   output.println("Changed first investment to RY");
```

```
 }
}
```

212

Further example usage

A Portfolio can be modified in strange ways

```
   // continued from previous slide
   for (Investment inv : ptf1)
      output.println(inv);
   output.println();
   Investment inv1 = ptf1.get(0);
   stk1 = inv1.getStock();
   stk1.setSymbol("RY");
   output.println("Changed first investment to RY");
   output.println(ptf1.toString());
    for (Investment inv : ptf1)
      output.println(inv);
 }
}
```

213

Further example usage

A Portfolio can be modified in strange ways: Test

```
% java PortfolioTest2
```

214

Further example usage

A Portfolio can be modified in strange ways: Test

```
% java PortfolioTest2
My e.g. Portfolio: 3
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99

Changed first investment to RY
My e.g. Portfolio: 3
RY Royal Bank of Canada QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
%
```

215

Further example usage

A Portfolio can be modified in strange ways: Test

```
% java PortfolioTest2
My e.g. Portfolio: 3
TD Toronto-Dominion Bank QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99

Changed first investment to RY
My e.g. Portfolio: 3
RY Royal Bank of Canada QTY=100 BV=81.32
BMO Bank of Montreal QTY=50 BV=60.15
RY Royal Bank of Canada QTY=100 BV=59.99
%
```

Remark

- Not only was the app able to change a Stock in an Investment,…
- …but it was able to leave behind an inconsistency between the Stock and the BV of the Investment.

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  {
```

217

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
```

218

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

219

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
```

220

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
```

221

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println("\nPortfolio ptf1 is:");
    output.println(ptf1.toString());
```

222

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println("\nPortfolio ptf1 is:");
    output.println(ptf1.toString());
   for (Investment inv : ptf1)
      output.println(inv);
```

223

Further example usage

Making a deep copy of a Portfolio

```
// assume the usual
import type.lib.*;
public class PortfolioDeepCopy
{ public static void main(String[ ] args)
  { Portfolio ptf1 = new Portfolio("My e.g. Portfolio", 10);
    Stock stk1 = new Stock("TD");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    stk1 = new Stock("BMO");
    ptf1.add(new Investment(stk1,50,stk1.getPrice()));
    stk1 = new Stock("RY");
    ptf1.add(new Investment(stk1,100,stk1.getPrice()));
    output.println("\nPortfolio ptf1 is:");
    output.println(ptf1.toString());
   for (Investment inv : ptf1)
      output.println(inv);
    // continued on next slide
```

224

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{



}
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{  Stock stkc = new Stock(inv.getStock().getSymbol());


}
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{  Stock stkc = new Stock(inv.getStock().getSymbol());
   Investment invc = new Investment(stkc, inv.getQty(),
                                    inv.getBookValue());

}
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{  Stock stkc = new Stock(inv.getStock().getSymbol());
   Investment invc = new Investment(stkc, inv.getQty(),
                                            inv.getBookValue());
   ptf1c.add(invc);
}
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{  Stock stkc = new Stock(inv.getStock().getSymbol());
   Investment invc = new Investment(stkc, inv.getQty(),
                                            inv.getBookValue());
   ptf1c.add(invc);
}
output.println("\nMade deep copy ptf1c of ptf1:");
ooutput.println(ptf1c.toString());
for (Investment inv : ptf1c)
   output.println(inv);
```

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
// make deep copy
Portfolio ptf1c = new Portfolio("My e.g. Portfolio", 10);
for (Investment inv : ptf1)
{  Stock stkc = new Stock(inv.getStock().getSymbol());
   Investment invc = new Investment(stkc, inv.getQty(),
                                                    inv.getBookValue());
   ptf1c.add(invc);
}
output.println("\nMade deep copy ptf1c of ptf1:");
ooutput.println(ptf1c.toString());
for (Investment inv : ptf1c)
   output.println(inv);
// continued on next slide
```

233

Further example usage

Making a deep copy of a Portfolio

```
// continued from previous slide
```

```
 }
}
```

234

Further example usage

Making a deep copy of a Portfolio

```
    // continued from previous slide
    // Change first investment in ptf1 to RY
```

```
  }
}
```

235

Further example usage

Making a deep copy of a Portfolio

```
    // continued from previous slide
    // Change first investment in ptf1 to RY
    Investment inv1 = ptf1.get(0);
```

```
  }
}
```

236

Further example usage

Making a deep copy of a Portfolio

```
    // continued from previous slide
    // Change first investment in ptf1 to RY
    Investment inv1 = ptf1.get(0);
    stk1 = inv1.getStock();
```

```
  }
}
```


Further example usage

Making a deep copy of a Portfolio

```
    // continued from previous slide
    // Change first investment in ptf1 to RY
    Investment inv1 = ptf1.get(0);
    stk1 = inv1.getStock();
    stk1.setSymbol("RY");
```

```
  }
}
```

Further example usage

Making a deep copy of a Portfolio

```
   // continued from previous slide
   // Change first investment in ptf1 to RY
   Investment inv1 = ptf1.get(0);
   stk1 = inv1.getStock();
   stk1.setSymbol("RY");
   output.println("\nChanged first investment in ptf1 to RY");
   output.println("\nptf1 is now:");
   output.println(ptf1.toString());
   for (Investment inv : ptf1)
      output.println(inv);




 }
}
```

239

Further example usage

Making a deep copy of a Portfolio

```
   // continued from previous slide
   // Change first investment in ptf1 to RY
   Investment inv1 = ptf1.get(0);
   stk1 = inv1.getStock();
   stk1.setSymbol("RY");
   output.println("\nChanged first investment in ptf1 to RY");
   output.println("\nptf1 is now:");
   output.println(ptf1.toString());
   for (Investment inv : ptf1)
      output.println(inv);
   output.println("\nptf1c is now:");
   output.println(ptf1c.toString());
   for (Investment inv : ptf1c)
      output.println(inv);
 }
}
```

240

Further example usage

Making a deep copy of a Portfolio: Test

% java PortfolioDeepCopy

241

Further example usage

Making a deep copy of a Portfolio: Test

% java PortfolioDeepCopy

- Try this out for yourself.
- Make sure you understand the results…
- …and the computations that led to them.

242

Outline

- Introduction
- Aggregation API
- Collections
- Algorithm complexity
- Further example usage
- **Appendix: Java class Vector**

243

Java class Vector

Recall arrays

- In retrospect, we recognize that the array provides us with a way to hold collections.
  - Indeed, arrays provide a generic collection mechanism in that we can specify the type held.

244

Java class Vector

Recall arrays

- In retrospect, we recognize that the array provides us with a way to hold collections.
  - Indeed, arrays provide a generic collection mechanism in that we can specify the type held.
- Further, we recall that once an array has been constructed in Java (necessarily with a specific length), its length stays fixed from then on.
  - Apparently, arrays (in Java) provide an example of static allocation (once they have been created).

245

Java class Vector

Recall arrays

- In retrospect, we recognize that the array provides us with a way to hold collections.
  - Indeed, arrays provide a generic collection mechanism in that we can specify the type held.
- Further, we recall that once an array has been constructed in Java (necessarily with a specific length), its length stays fixed from then on.
  - Apparently, arrays (in Java) provide an example of static allocation (once they have been created).
- If you want something akin to an array that can be resized after creation, then Java provides the Vector class.

246

Vectors

Basics

- A Vector is a container for *objects* that grows automatically.
  - Here we see another Java provided approach to dealing with collections.
  - In this case, we encounter dynamic allocation (as opposed to the static allocation of arrays).

247

Vectors

Basics

- A Vector is a container for *objects* that grows automatically.
  - Here we see another Java provided approach to dealing with collections.
  - In this case, we encounter dynamic allocation (as opposed to the static allocation of arrays).
- The Vector class is in java.util., hence we must explicitly import it into out programs

  import java.util.Vector;

248

Vectors

Basics

- A Vector is a container for *objects* that grows automatically.
  - Here we see another Java provided approach to dealing with collections.
  - In this case, we encounter dynamic allocation (as opposed to the static allocation of arrays).
- The Vector class is in java.util., hence we must explicitly import it into out programs

  import java.util.Vector;
- We declare and construct an instance of a Vector object in the following fashion

  Vector studNames = new Vector();

249

Vectors

Basics

- Given declaration and construction

  Vector studNames = new Vector();
- Subsequently we can insert elements with the add method.

```
while (true)
{ output.print("Enter a student name (return to end): ");
   String s = input.nextLine();
   if (s.equals(""))
     break;
   else
     studNames.add(s); // adds to end of the vector
 }
```

250

Vectors

Basics

- We can set the value at an arbitrary position in a vector via the set method.

  studNames.set(1, “Steve”); // position 1 now has Steve

2

251

Vectors

Basics

- We can set (i.e., replace) the value at an arbitrary position in a vector via the set method.

  studNames.set(1, “Steve”); // position 1 now has Steve

- We also can insert an object in the middle of a vector, while moving all the other elements down by one position.

  studNames.add(1, “Eshrat”); // position 1 has Eshrat;
  // whatever was at 1 now at 2

252

Vectors

Basics

- We can set (i.e., replace) the value at an arbitrary position in a vector via the set method.

  studNames.set(1, “Steve”); // position 1 now has Steve

- We also can insert an object in the middle of a vector, while moving all the other elements down by one position.

  studNames.add(1, “Eshrat”); // position 1 has Eshrat;
  // whatever was at 1 now at 2

- We can access the number of elements in a vector with the size method.

  int n = studNames.size();

253

Vectors

Basics

- Getting data back out of a Vector is a bit more involved.
- A Vector can hold *objects* of any type.
- Indeed, when an object is inserted into a Vector, its reference is automatically converted to that of type Object, the Java abstraction of all other objects.

254

Vectors

Basics

- Getting data back out of a Vector is a bit more involved.
- A Vector can hold *objects* of any type.
- Indeed, when an object is inserted into a Vector, its reference is automatically converted to that of type Object, the Java abstraction of all other objects.
- When reading an object back out, we use the get method; typically we cast to the object's original type

```
for (int j=0; j<studNames.size(); j++)
{ String s = (String) studNames.get(j);
  ...
}
```

255

Vectors

Basics

- Getting data back out of a Vector is a bit more involved.
- A Vector can hold *objects* of any type.
- Indeed, when an object is inserted into a Vector, its reference is automatically converted to that of type Object, the Java abstraction of all other objects.
- When reading an object back out, we use the get method; typically we cast to the object's original type

```
for (int j=0; j<studNames.size(); j++)
{ String s = (String) studNames.get(j);
  ...
}
```

- Remember: If in doubt about what you are about to cast, then use instanceof.

256

Vectors

Storing primitive types in a Vector

- We can store an *object* of any class in a Vector.
- In contrast, *primitive types* (integers, floating-point numbers, truth values) cannot be stored in a Vector directly as they are not objects.

257

Vectors

Storing primitive types in a Vector

- We can store an *object* of any class in a Vector.
- In contrast, *primitive types* (integers, floating-point numbers, truth values) cannot be stored in a Vector directly as they are not objects.
- We resort to the notion of wrapper classes.
  - The classes Integer, Double, Boolean, etc. can be used to wrap numbers and truth values inside appropriate classes.
- Let's see how this is done…

258

Vectors

Storing primitive types in a Vector

- Example: Getting a floating point number into a Vector

```
Double aMark = new Double(75.57);
Vector marks = new Vector();
marks.add(aMark);
```

Vectors

Storing primitive types in a Vector

- Example: Getting a floating point number into a Vector

```
Double aMark = new Double(75.57);
Vector marks = new Vector();
marks.add(aMark);
```

alternatively

```
Vector marks = new Vector();
marks.add(new Double(75.57));
```

Vectors

Storing primitive types in a Vector

- Example: Getting a floating point number into a Vector

```
Double aMark = new Double(75.57);
Vector marks = new Vector();
marks.add(aMark);
```

alternatively

```
Vector marks = new Vector();
marks.add(new Double(75.57));
```

- Example: Getting a floating point number out of a Vector

```
double aMarkPrim = ((Double)marks.get(0)).doubleValue();
```

Notice that here we need to

1. Access via get method of Vector
2. Cast to Double
3. Invoke the doubleValue method of Double

261

Vectors

Storing primitive types in a Vector

- Example: Getting a floating point number into a Vector

```
Double aMark = new Double(75.57);
Vector marks = new Vector();
marks.add(aMark);
```

alternatively

```
Vector marks = new Vector();
marks.add(new Double(75.57));
```

- Example: Getting a floating point number out of a Vector

```
double aMarkPrim = ((Double)marks.get(0)).doubleValue();
```

Notice that here we need to

1. Access via get method of Vector
2. Cast to Double
3. Invoke the doubleValue method of Double

Remark: Yes this is a hassle!

262

Vectors

Storing primitive types in a Vector

- Example: Getting an integer number into a Vector

```
Integer aNum = new Integer(12);
Vector nums = new Vector();
nums.add(aNum);
```

alternatively

```
Vector nums = new Vector();
nums.add(new Integer(12));
```

263

Vectors

Storing primitive types in a Vector

- Example: Getting an integer number into a Vector

```
Integer aNum = new Integer(12);
Vector nums = new Vector();
nums.add(aNum);
```

alternatively

```
Vector nums = new Vector();
nums.add(new Integer(12));
```

- Example: Getting an integer number out of a Vector

```
int aNumPrim = ((Integer)nums.get(0)).intValue();
```

Notice that here we need to

1. Access via get method of Vector
2. Cast to Integer
3. Invoke the intValue method of Integer

264

Vectors

Converting vectors to arrays

- Because of the overhead in getting data out of a vector, we often
  - Use a vector for reading in a data set of unknown size
  - Convert the vector to an array for subsequent data processing.

265

Vectors

Converting vectors to arrays

- Because of the overhead in getting data out of a vector, we often
  - Use a vector for reading in a data set of unknown size
  - Convert the vector to an array for subsequent data processing.
    1. Create a new array with length equal to the size of the vector.
    2. Copy the vector elements into the array elements.

266

Vectors

Converting vectors to arrays

- Because of the overhead in getting data out of a vector, we often
  - Use a vector for reading in a data set of unknown size
  - Convert the vector to an array for subsequent data processing.
    1. Create a new array with length equal to the size of the vector.
    2. Copy the vector elements into the array elements.
- This approach allows us to combine the strength of vectors (dynamic growth) with the strength of an array (ease of access to individual elements).
- Let's see how this is done…

267

Vectors

Converting vectors to arrays

```
String s;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a student name (return to end): ");
   s = input.nextLine();
    if (s.equals(""))
       break; // this is the way out of the loop
    else
       inputVector.add(s);
}
```

268

Vectors

Converting vectors to arrays

```
String s;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a student name (return to end): ");
  s = input.nextLine();
   if (s.equals(""))
      break; // this is the way out of the loop
   else
       inputVector.add(s);
}
String[ ]  studNames = new String[inputVector.size()];
```

269

Vectors

Converting vectors to arrays

```
String s;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a student name (return to end): ");
  s = input.nextLine();
   if (s.equals(""))
      break; // this is the way out of the loop
   else
       inputVector.add(s);
}
String[ ]  studNames = new String[inputVector.size()];
inputVector.copyInto(studNames); // copies vector to array
```

270

Vectors

Example: Improvements to MarksAnalysis

- We will make use of the Vector class to allow for arbitrary amounts of input, …
- …without the user telling in advance how much input to expect.

```
public class MarksAnalysis
{ public static void main(String[ ] args)
  { // declaration
    // input
    // computation
    // output
  }
}
```

271

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
```

272

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
```

273

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{



}
```

274

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();


}
```

275

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop

}
```

276

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
```

277

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
if (inputVector.size() != 0)
{



}
```

278

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
if (inputVector.size() != 0)
{ marks = new double[inputVector.size()];


}
```

279

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
if (inputVector.size() != 0)
{ marks = new double[inputVector.size()];
   for (int j=0; j<marks.length; j++)
       marks[j] = ((Double) inputVector.get(j)).doubleValue();
}
```

280

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
if (inputVector.size() != 0)
{ marks = new double[inputVector.size()];
   for (int j=0; j<marks.length; j++)
      marks[j] = ((Double) inputVector.get(j)).doubleValue();
}
```

Remark: We cannot use the Vector method copyInto as the array we copying to has primitive type elements, double.

281

Vectors

Example: Improvements to MarksAnalysis

```
// declaration and input
double mark;
double[ ] marks = null;
Vector inputVector = new Vector();
while (true)
{ output.print("Enter a mark (negative to stop): ");
  mark = input.nextDouble();
  if (mark < 0) break; // this is the way out of the loop
  inputVector.add(new Double(mark));
}
if (inputVector.size() != 0)
{ marks = new double[inputVector.size()];
   for (int j=0; j<marks.length; j++)
      marks[j] = ((Double) inputVector.get(j)).doubleValue();
}
```

Remark:

- We see that input flexibility has come at the cost of more complicated code.
- This is typical.

282

Vectors

Example: Improvements to MarksAnalysis

```
// computation and output
```

283

Vectors

Example: Improvements to MarksAnalysis

```
// computation and output
if (marks == null)
  output.println("No marks entered.");
```

284

Vectors

Example: Improvements to MarksAnalysis

```
// computation and output
if (marks == null)
  output.println("No marks entered.");
else
{




}
```

285

Vectors

Example: Improvements to MarksAnalysis

```
// computation and output
if (marks == null)
  output.println("No marks entered.");
else
{ double sum = 0;
  for (int j=0; j< marks.length; j++)
     sum += marks[j];

}
```

286

Vectors

Example: Improvements to MarksAnalysis

```
// computation and output
if (marks == null)
   output.println("No marks entered.");
else
{ double sum = 0;
  for (int j=0; j< marks.length; j++)
     sum += marks[j];
  output.println("The class average is " + sum / marks.length);
}
```

287

Vectors

Example: Improvements to MarksAnalysis

```
public class MarksAnalysis
{ public static void main(String[ ] args)
  { // declaration
    // input
    // computation
    // output
  }
}
```

Remark: Implementation complete.

288

Summary

- Introduction
- Aggregation API
- Collections
- Algorithm complexity
- Further example usage
- Appendix: Java class Vector

289