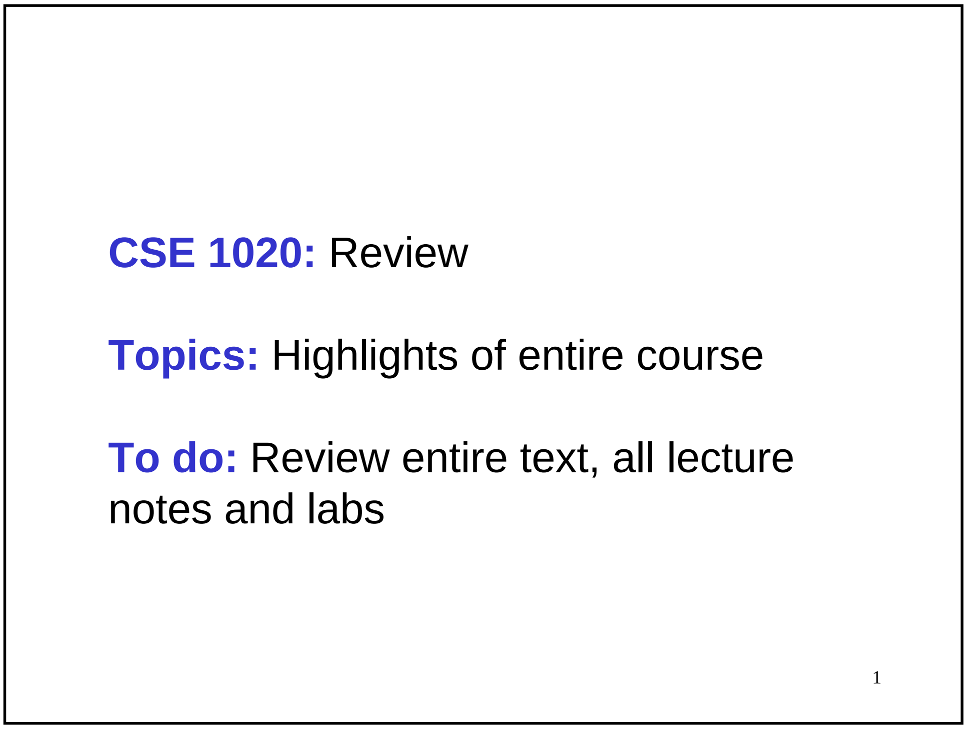
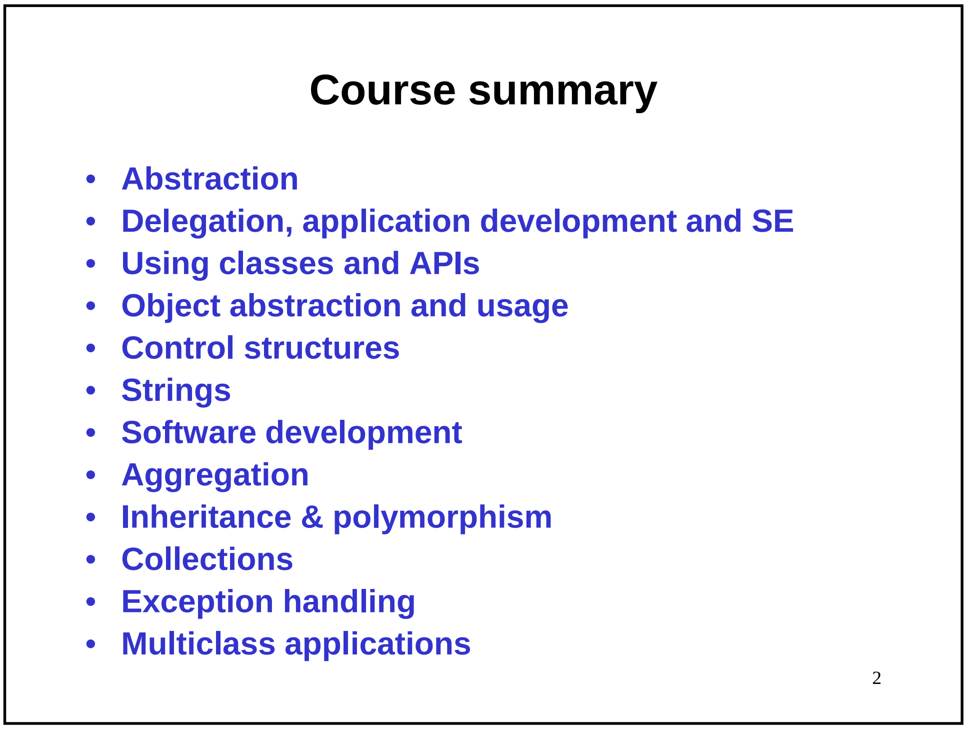


**CSE 1020:** Review

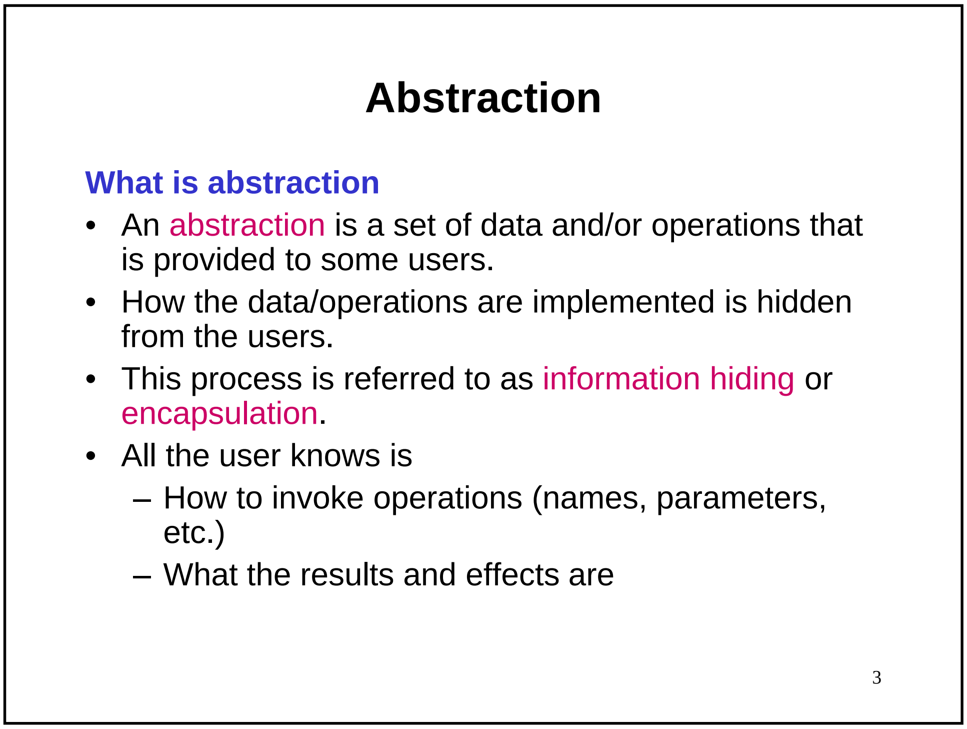
**Topics:** Highlights of entire course

**To do:** Review entire text, all lecture notes and labs



## Course summary

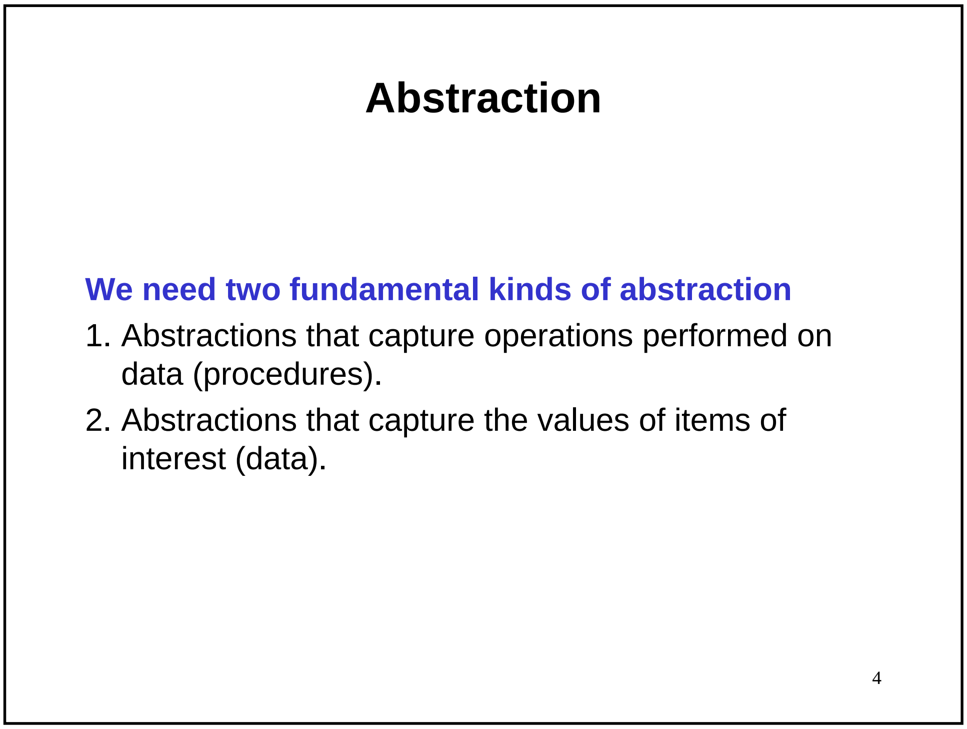
- **Abstraction**
- **Delegation, application development and SE**
- **Using classes and APIs**
- **Object abstraction and usage**
- **Control structures**
- **Strings**
- **Software development**
- **Aggregation**
- **Inheritance & polymorphism**
- **Collections**
- **Exception handling**
- **Multiclass applications**



# Abstraction

**What is abstraction**

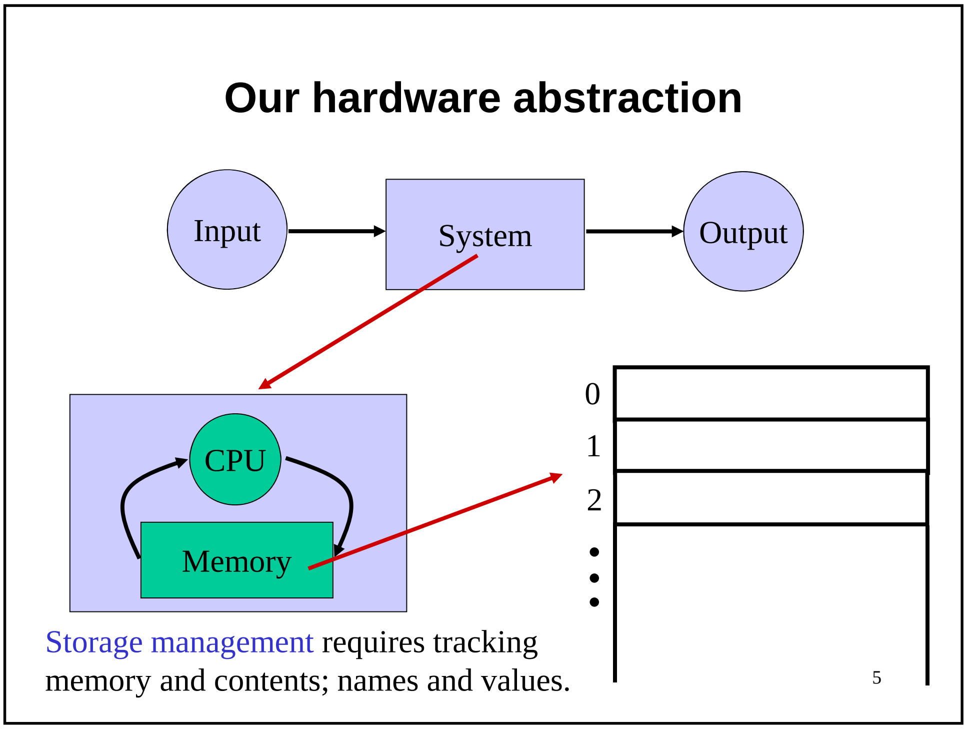
- An abstraction is a set of data and/or operations that is provided to some users.
- How the data/operations are implemented is hidden from the users.
- This process is referred to as information hiding or encapsulation.
- All the user knows is
  - How to invoke operations (names, parameters, etc.)
  - What the results and effects are



# Abstraction

**We need two fundamental kinds of abstraction**

1. Abstractions that capture operations performed on data (procedures).
2. Abstractions that capture the values of items of interest (data).



## Our hardware abstraction

Input

System

Output

CPU

Memory

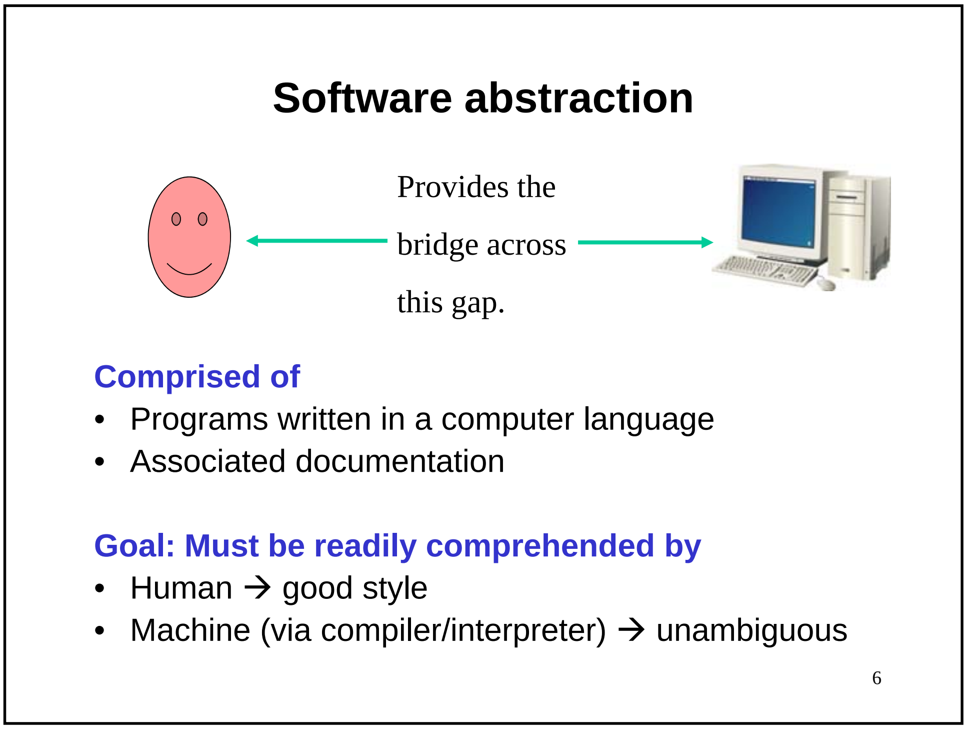
0

1

2

•
•
•

Storage management requires tracking memory and contents; names and values.



## Software abstraction

Provides the

bridge across

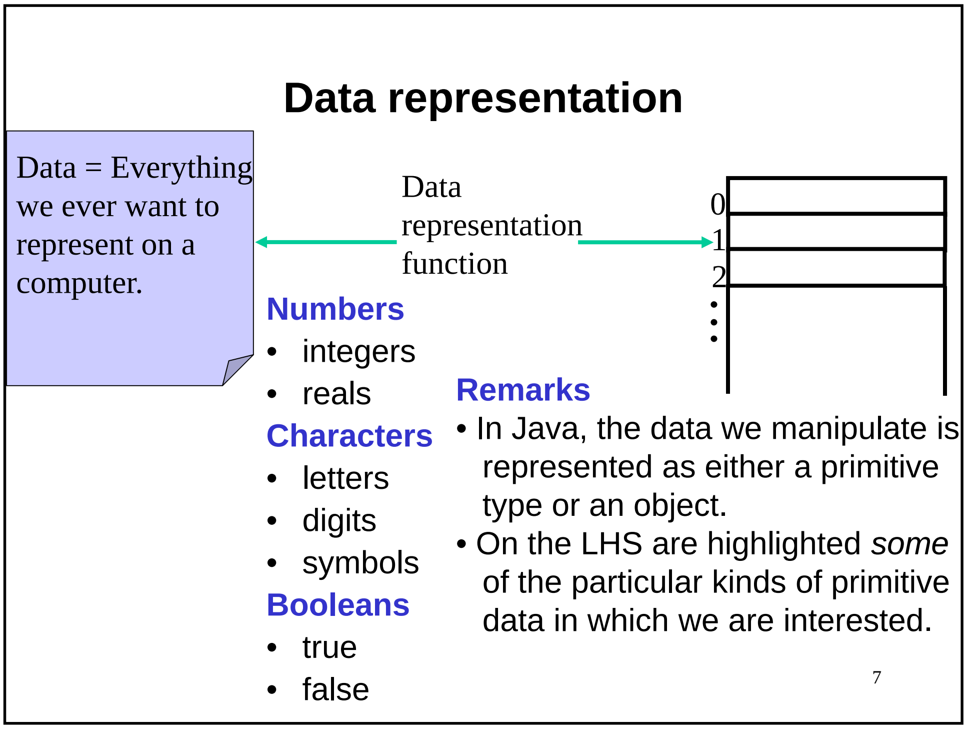
this gap.

**Comprised of**

- Programs written in a computer language
- Associated documentation

**Goal: Must be readily comprehended by**

- Human → good style
- Machine (via compiler/interpreter) → unambiguous



## Data representation

Data = Everything we ever want to represent on a computer.

Data representation function

0
1
2
⋮

**Numbers**

- integers
- reals

**Characters**

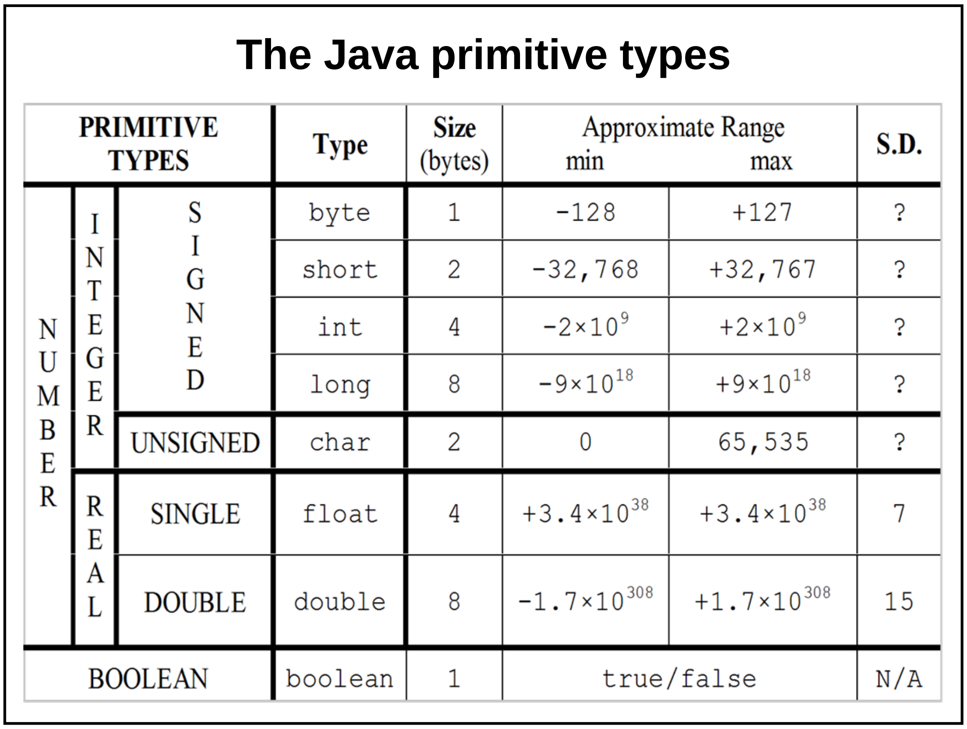
- letters
- digits
- symbols

**Booleans**

- true
- false

**Remarks**

- In Java, the data we manipulate is represented as either a primitive type or an object.
- On the LHS are highlighted *some* of the particular kinds of primitive data in which we are interested.



## The Java primitive types

| PRIMITIVE TYPES | | | Type | Size (bytes) | Approximate Range min | max | S.D. |
|---|---|---|---|---|---|---|---|
| NUMBER | INTEGER | SIGNED | byte | 1 | -128 | +127 | ? |
| | | | short | 2 | -32,768 | +32,767 | ? |
| | | | int | 4 | $-2\times10^{9}$ | $+2\times10^{9}$ | ? |
| | | | long | 8 | $-9\times10^{18}$ | $+9\times10^{18}$ | ? |
| | | UNSIGNED | char | 2 | 0 | 65,535 | ? |
| | REAL | SINGLE | float | 4 | $+3.4\times10^{38}$ | $+3.4\times10^{38}$ | 7 |
| | | DOUBLE | double | 8 | $-1.7\times10^{308}$ | $+1.7\times10^{308}$ | 15 |
| BOOLEAN | | | boolean | 1 | true/false | | N/A |

## Variables

### What

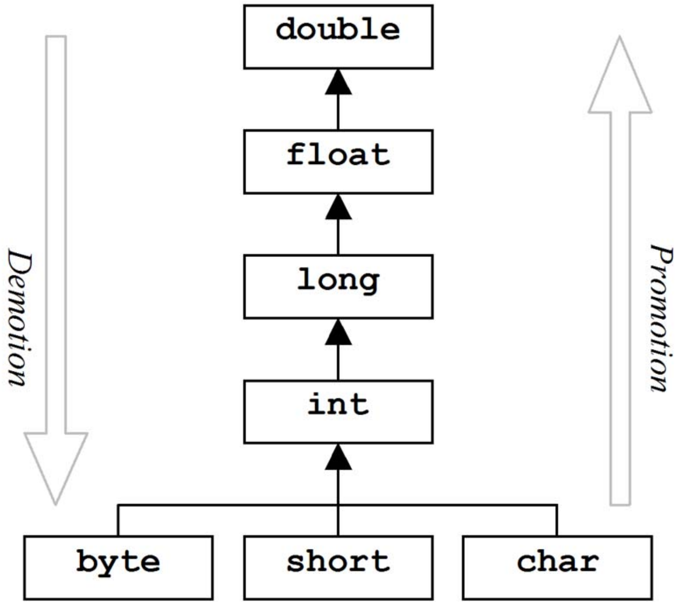
- Variables are entities in a program that have a value which is allowed to change during the course of the program.

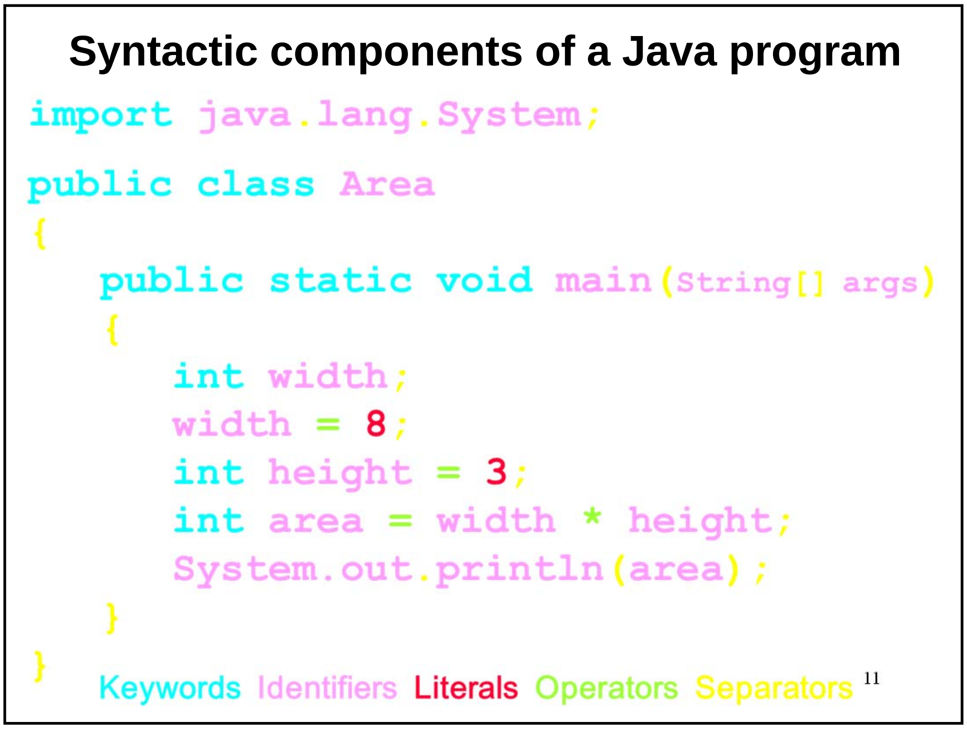
```
int amount, numQuarters;
amount = 78;
numQuarters = amount / 25;
amount = amount – numQuarters * 25;
```

### Why

- Variables provide a way to model items with values that vary during the time we interested in them.
- They allow us to abstract away from details of machine representation.

## Type promotion and casting



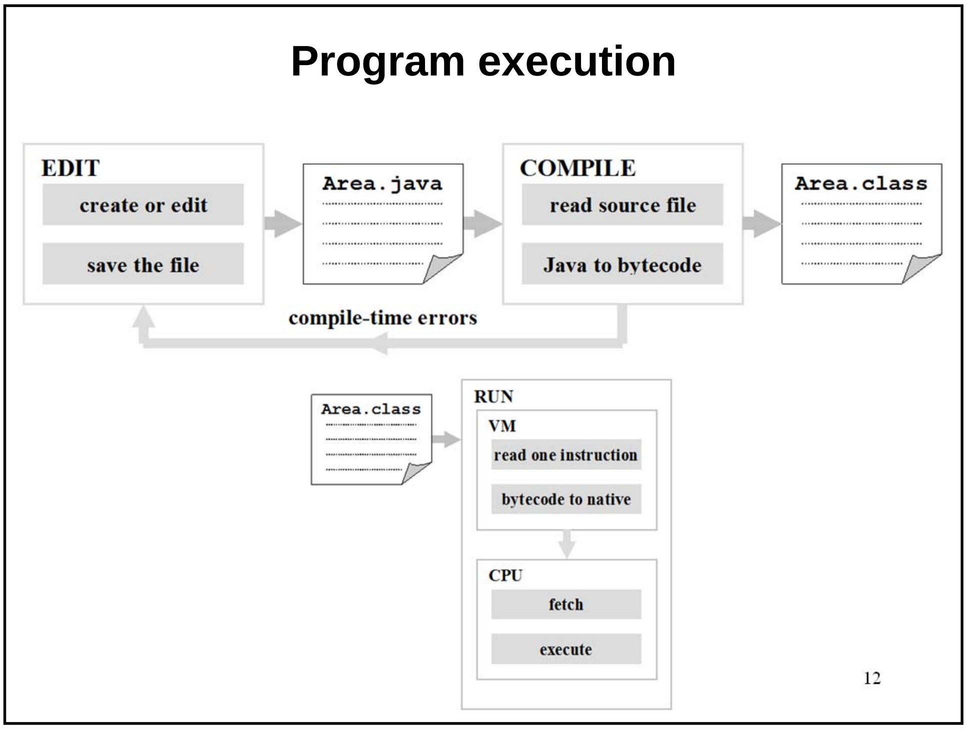


## Syntactic components of a Java program

```
import java.lang.System;

public class Area
{
   public static void main(String[] args)
   {
      int width;
      width = 8;
      int height = 3;
      int area = width * height;
      System.out.println(area);
   }
}
```

Keywords Identifiers Literals Operators Separators



## Program execution

EDIT

create or edit

save the file

Area.java

COMPILE

read source file

Java to bytecode

Area.class

compile-time errors

Area.class

RUN

VM

read one instruction

bytecode to native

CPU

fetch

execute

# Delegation

## What

- Delegation is an abstraction strategy that allows us to deal with the complexity inherent in large systems.
- We delegate parts of the task to other mechanisms.
- We consider two ways to delegate:

1. Delegation to a static method
2. Delegation to an object

# Delegation

## Delegation to a static method

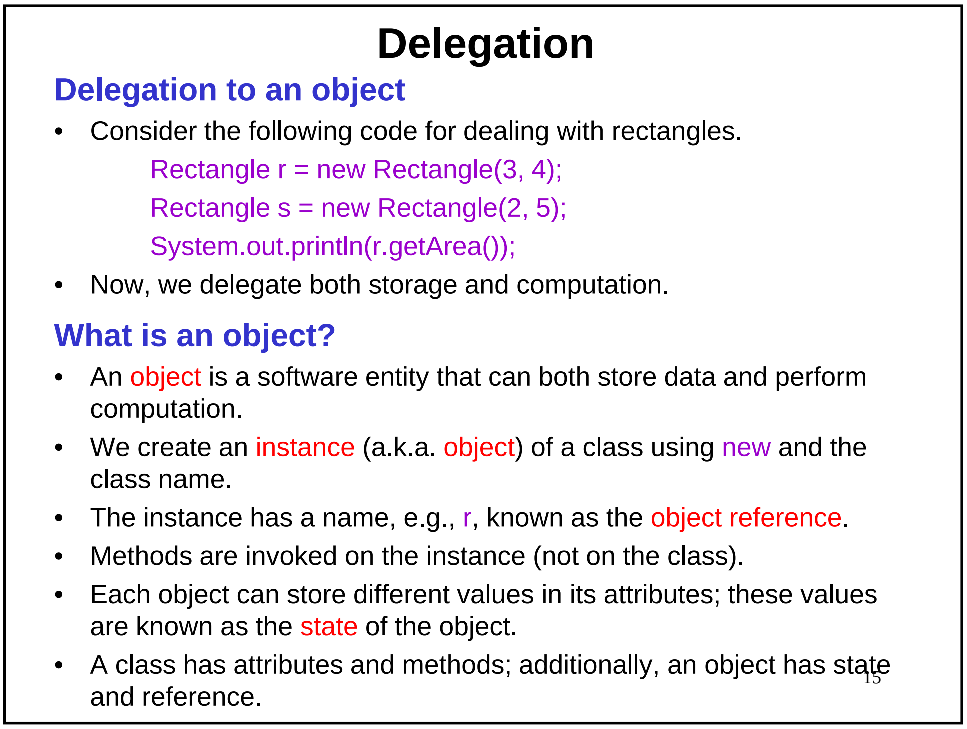
- Consider the following code for obtaining Body Mass Index (BMI).

```
double weight = 165.0;
String height = “6’1”;
double bmi = ToolBox.getBMI(weight, height);
```

- We maintain our own our own storage, but …
- … delegate the computation to a class.

## What do we mean by “static method”?

- A method performs an action.
  - Its name (typically) is a verb (getBMI) or a predicate (isEnabled).
- Methods belong to classes.
- The invocation syntax is class_name.method(…).
  - With the method’s parameters (if any) substituted for “…”.
- Methods terminate with a return, which might be void.
- The keyword static notes that the method neither inspects nor modifies class copies. (Look back to Unit 1!)



# Delegation

## Delegation to an object

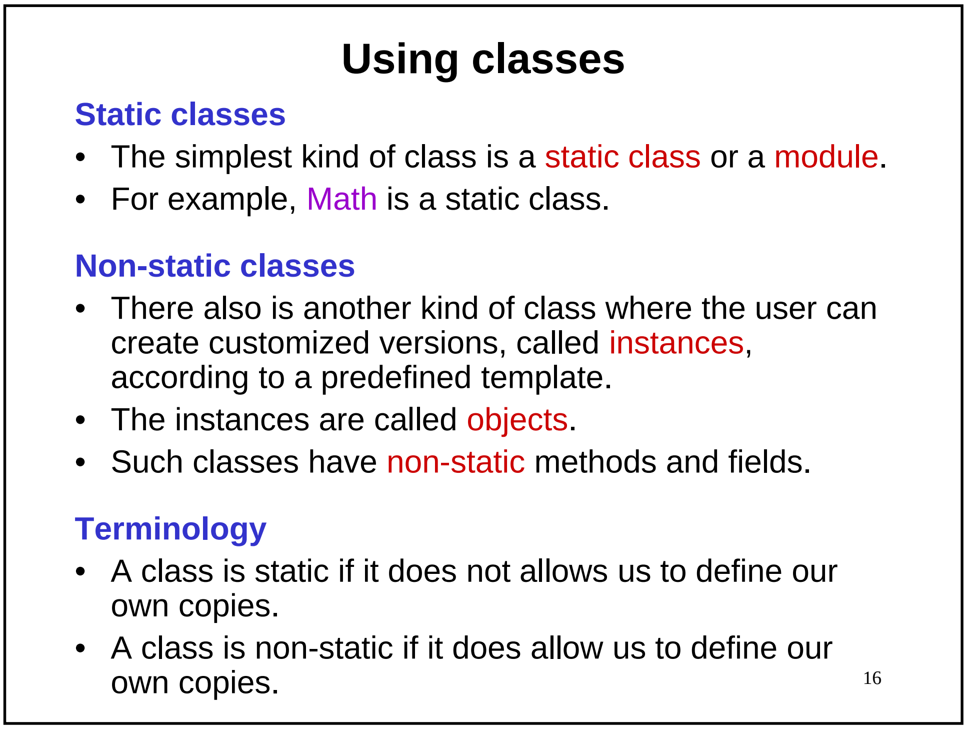
- Consider the following code for dealing with rectangles.

```
Rectangle r = new Rectangle(3, 4);
Rectangle s = new Rectangle(2, 5);
System.out.println(r.getArea());
```

- Now, we delegate both storage and computation.

## What is an object?

- An object is a software entity that can both store data and perform computation.
- We create an instance (a.k.a. object) of a class using new and the class name.
- The instance has a name, e.g., r, known as the object reference.
- Methods are invoked on the instance (not on the class).
- Each object can store different values in its attributes; these values are known as the state of the object.
- A class has attributes and methods; additionally, an object has state and reference.



# Using classes

## Static classes

- The simplest kind of class is a static class or a module.
- For example, Math is a static class.

## Non-static classes

- There also is another kind of class where the user can create customized versions, called instances, according to a predefined template.
- The instances are called objects.
- Such classes have non-static methods and fields.

## Terminology

- A class is static if it does not allows us to define our own copies.
- A class is non-static if it does allow us to define our own copies.

# Using classes

| Non-static classes can have | Static classes can have |
|---|---|
| constructors | |
| instance (non-static) methods | |
| instance (non-static) attributes | |
| static methods | static methods |
| static attributes | static attributes |

# APIs

## What is an API

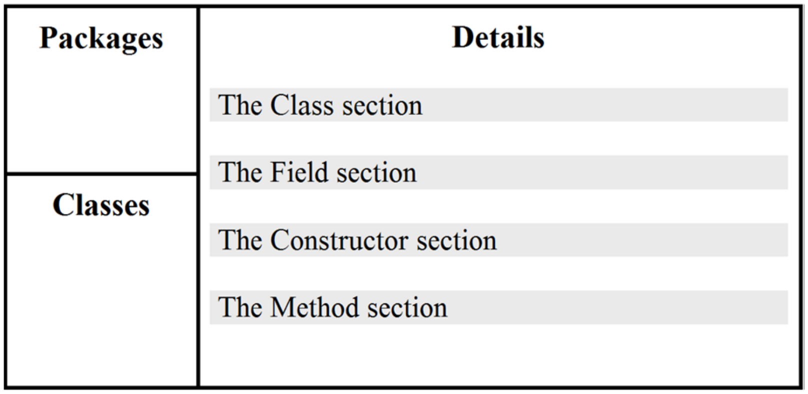
- The term API stands for Application Programming Interface
- Documents how another program can access a given class.
- Hides implementation detail.

## Why we care: Guide to ready made software modules

- As an applications programmer, we use the API of a class for two main reasons
  1. By perusing the API of a class we can determine if it provides useful functionality for the task that we are addressing.
  2. If we discover useful functionality, then the API tells us how to access it.

# APIs

## API anatomy: Overall layout



Packages

Classes

Details

The Class section

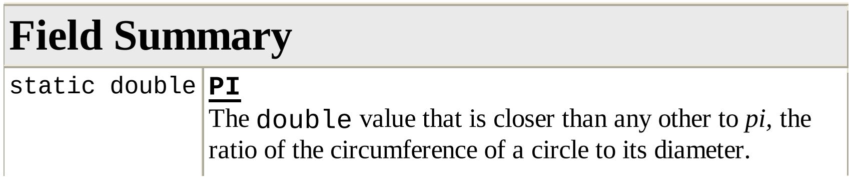
The Field section

The Constructor section

The Method section

# APIs

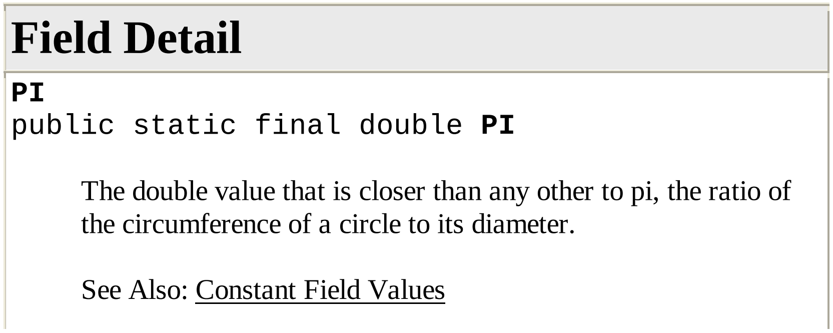
## API anatomy: Fields



| Field Summary | |
|---|---|
| `static double` | **`PI`**<br>The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter. |

# APIs

## API anatomy: Fields



### Field Detail

**PI**

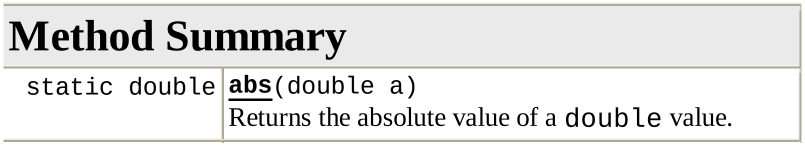
```
public static final double PI
```

The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.

See Also: Constant Field Values

# APIs

## API anatomy: Methods



### Method Summary

| | |
|---|---|
| `static double` | **`abs`**`(double a)`<br>Returns the absolute value of a `double` value. |

# APIs

## API anatomy: Methods

### Method Detail

**`abs`**

**`public static double abs(double a)`**

Returns the absolute value of a `double` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is `NaN`, the result is `NaN`.

**Parameters:**
`a` - the argument whose absolute value is to be determined

**Returns:**
the absolute value of the argument.

# APIs

## API anatomy: Constructors

**Constructor Summary**

Stock()
Construct a default Stock.

Stock(Stock stock)
Construct a copy of the passed Stock.

Stock(java.lang.String symbol)
Construct a Stock having the (capitalized) passed symbol.

# APIs

## API anatomy: Constructors

**Constructor Detail**

Stock

public Stock(java.lang.String symbol)

Construct a Stock having the (capitalized) passed symbol. The stock attributes are set as per the refresh() method.

Parameters:

symbol - the (ticker) symbol of the stock to construct.

Stock

public Stock(Stock stock)

Construct a copy of the passed stock.

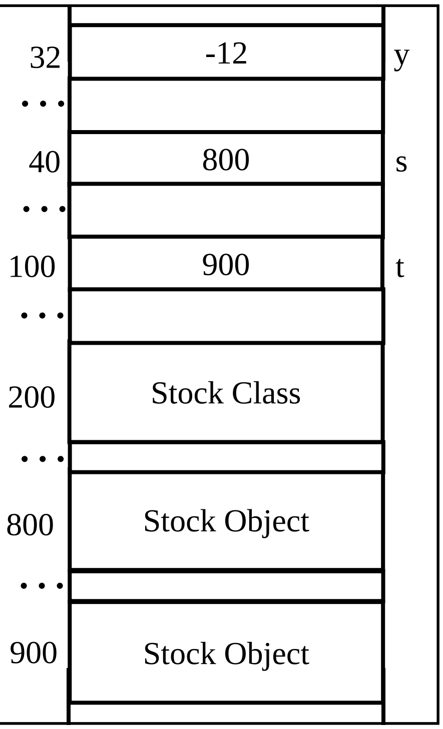
Parameters: stock – the Stock to copy.

Throws: java.lang.RuntimeException –if Stock is null.

# Class usage

## Object vs. object reference

```
import type.lib.*;
int y;
y = -12;
Stock s;
s = new Stock(“RY”);
Stock t = s;
assert s ==  t; // assert okay
t = null;
assert s == t; // assert fails
t = new Stock(“RY”);
assert s == t; // assert fails
```



32 | -12 | y

• • •

40 | 800 | s

• • •

100 | 900 | t

• • •

200 | Stock Class

• • •

800 | Stock Object

• • •

900 | Stock Object

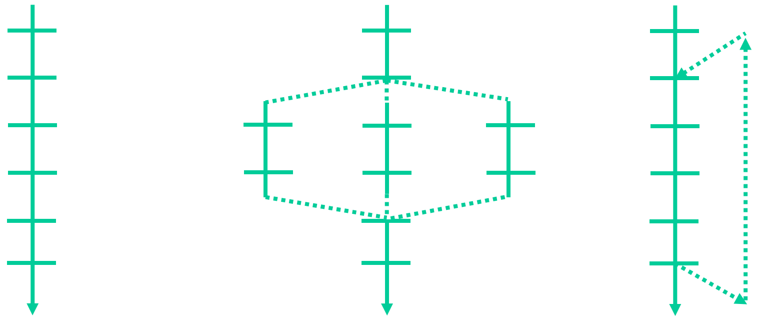
# Abstraction: Objects, classes & methods

**What we now understand**

- Object oriented programming supports modular software development and high reuse.
- Classes provide data abstraction and modular design.
- Objects are instances of classes.
- Methods provide procedural abstraction.

# Control structures

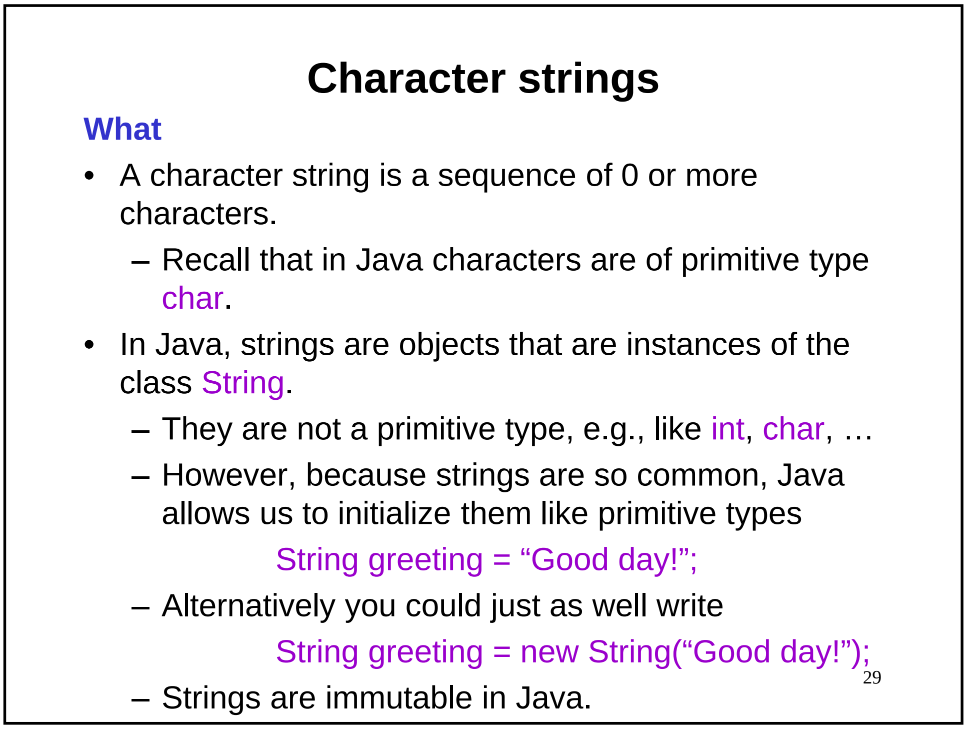
**We have now seen three types of control structure**



**Sequence:** straight line code

**Selection:** if; switch

**Iteration:** for; while; do



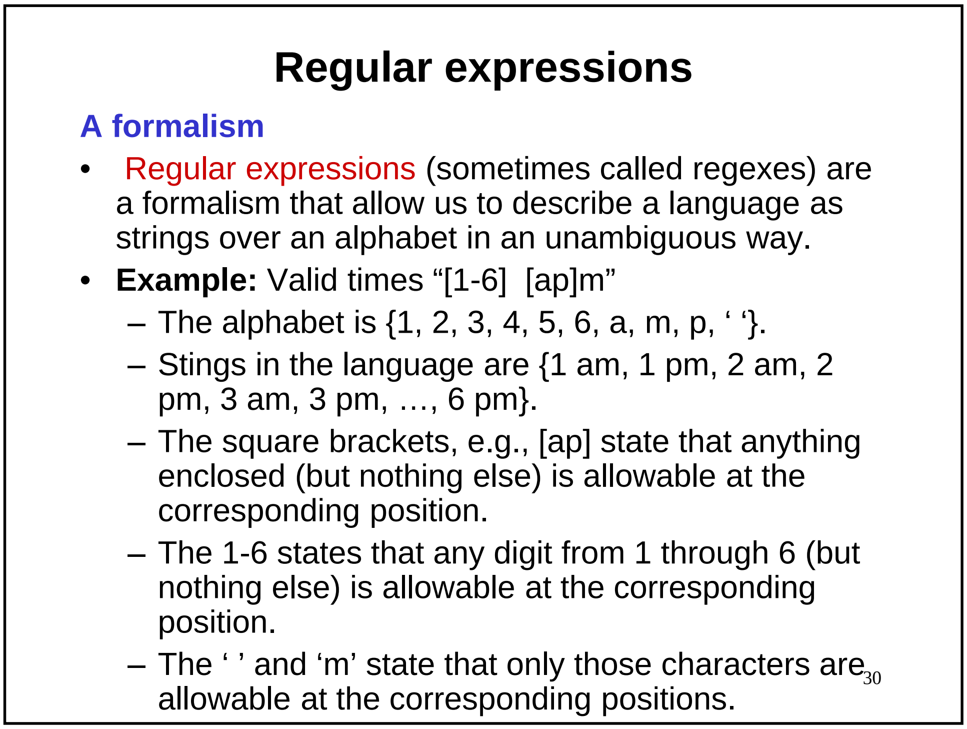
## Character strings

### What

- A character string is a sequence of 0 or more characters.
  - Recall that in Java characters are of primitive type char.
- In Java, strings are objects that are instances of the class String.
  - They are not a primitive type, e.g., like int, char, …
  - However, because strings are so common, Java allows us to initialize them like primitive types

    String greeting = “Good day!”;
  - Alternatively you could just as well write

    String greeting = new String(“Good day!”);
  - Strings are immutable in Java.



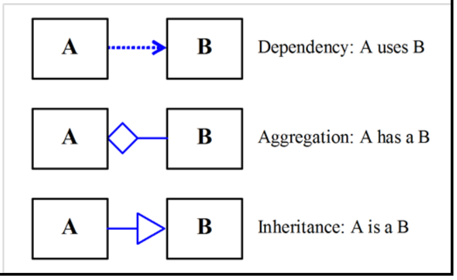
## Regular expressions

### A formalism

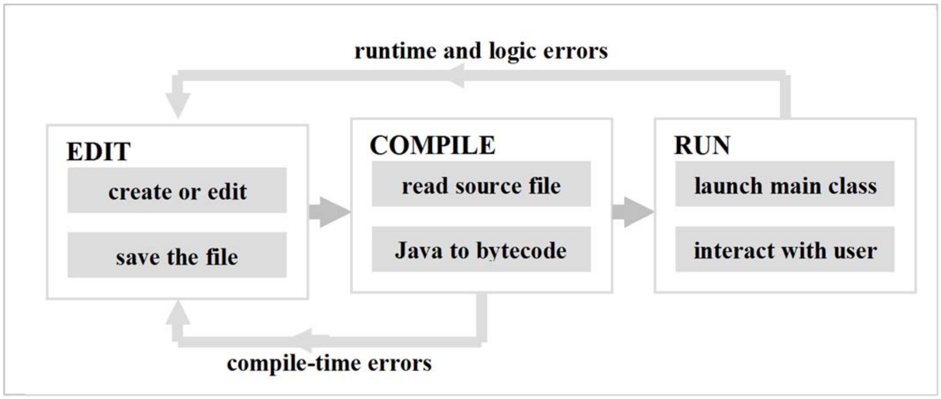
- Regular expressions (sometimes called regexes) are a formalism that allow us to describe a language as strings over an alphabet in an unambiguous way.
- **Example:** Valid times “[1-6]  [ap]m”
  - The alphabet is {1, 2, 3, 4, 5, 6, a, m, p, ‘ ‘}.
  - Stings in the language are {1 am, 1 pm, 2 am, 2 pm, 3 am, 3 pm, …, 6 pm}.
  - The square brackets, e.g., [ap] state that anything enclosed (but nothing else) is allowable at the corresponding position.
  - The 1-6 states that any digit from 1 through 6 (but nothing else) is allowable at the corresponding position.
  - The ‘ ’ and ‘m’ state that only those characters are allowable at the corresponding positions.

# Software engineering and development

- **Waterfall vs. iterative model**
- **Testing**
  - – Black box vs. white box
  - – Test harness
- **Unified Modeling Language (UML)**
  - – A formal visual language for depicting classes and their interrelationships.



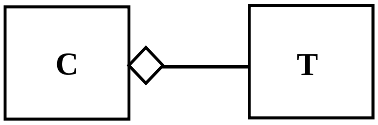
# Edit/compile/run & errors



# Aggregation

## What

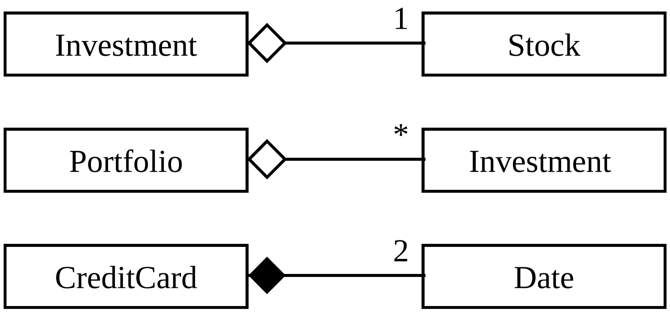
- A typical software system uses several classes, including the app.
- It is useful to depict the interrelationships that hold.
- Aggregation (has-a): Class C aggregates class T if C has T as an attribute.
- We call C the aggregate class.
- We call T the component or aggregated class.



# Aggregation

## Multiplicity

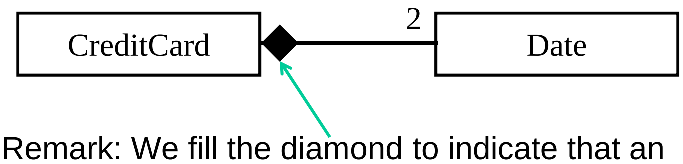
- In addition to the aggregate and aggregated classes, aggregation is characterized via multiplicity.
- Multiplicity is the number of attributes in the aggregate class that are of the aggregated type.



# Aggregation

## 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.

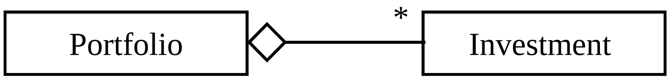


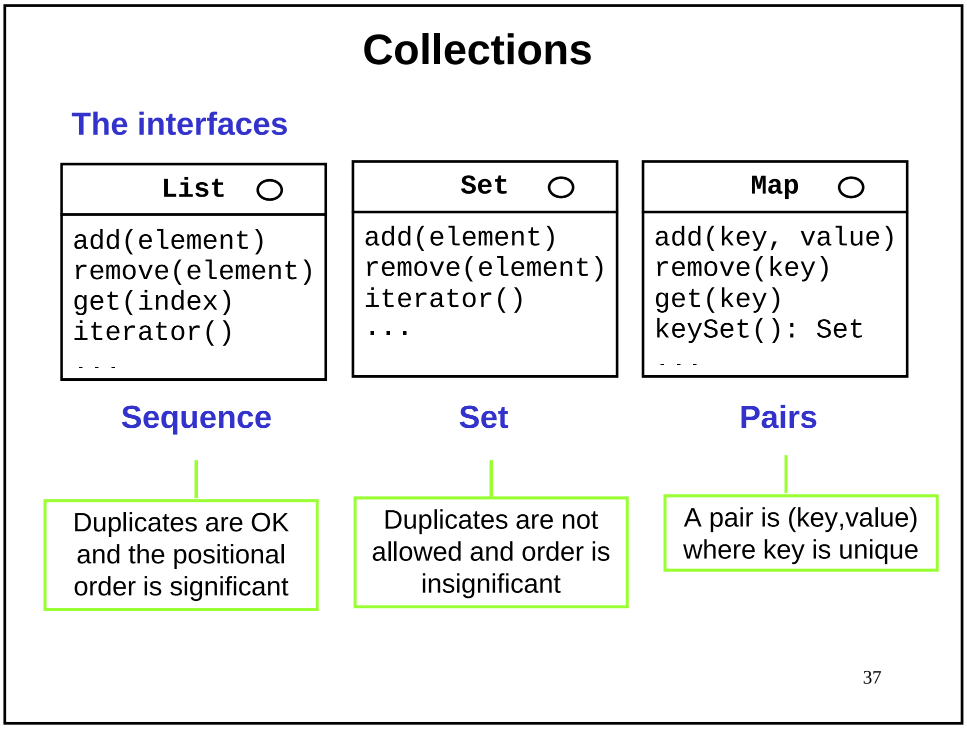
- Remark: We fill the diamond to indicate that an aggregate is a composition.

# Aggregation

## 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.





## Collections

### The interfaces

```
List ○
add(element)
remove(element)
get(index)
iterator()
...
```

**Sequence**

Duplicates are OK and the positional order is significant

```
Set ○
add(element)
remove(element)
iterator()
...
```

**Set**

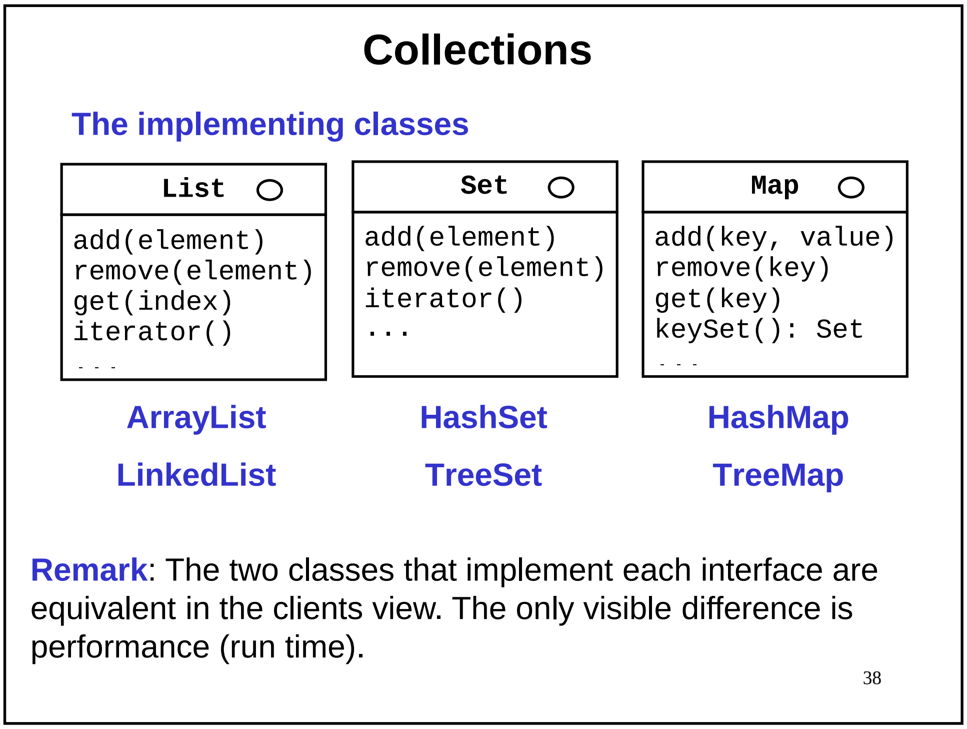
Duplicates are not allowed and order is insignificant

```
Map ○
add(key, value)
remove(key)
get(key)
keySet(): Set
...
```

**Pairs**

A pair is (key,value) where key is unique

37



## Collections

### The implementing classes

```
List ○
add(element)
remove(element)
get(index)
iterator()
...
```

**ArrayList**

**LinkedList**

```
Set ○
add(element)
remove(element)
iterator()
...
```

**HashSet**

**TreeSet**

```
Map ○
add(key, value)
remove(key)
get(key)
keySet(): Set
...
```

**HashMap**

**TreeMap**

**Remark**: The two classes that implement each interface are equivalent in the clients view. The only visible difference is performance (run time).

38

# Collections

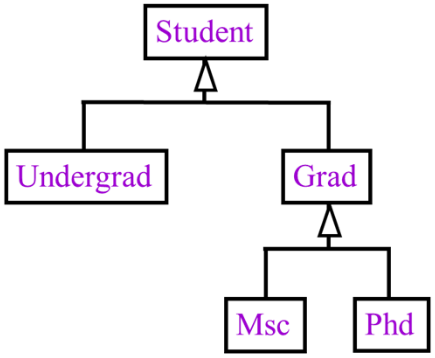
**Generics**

- Client-based strong typing
- Supported by the collections framework

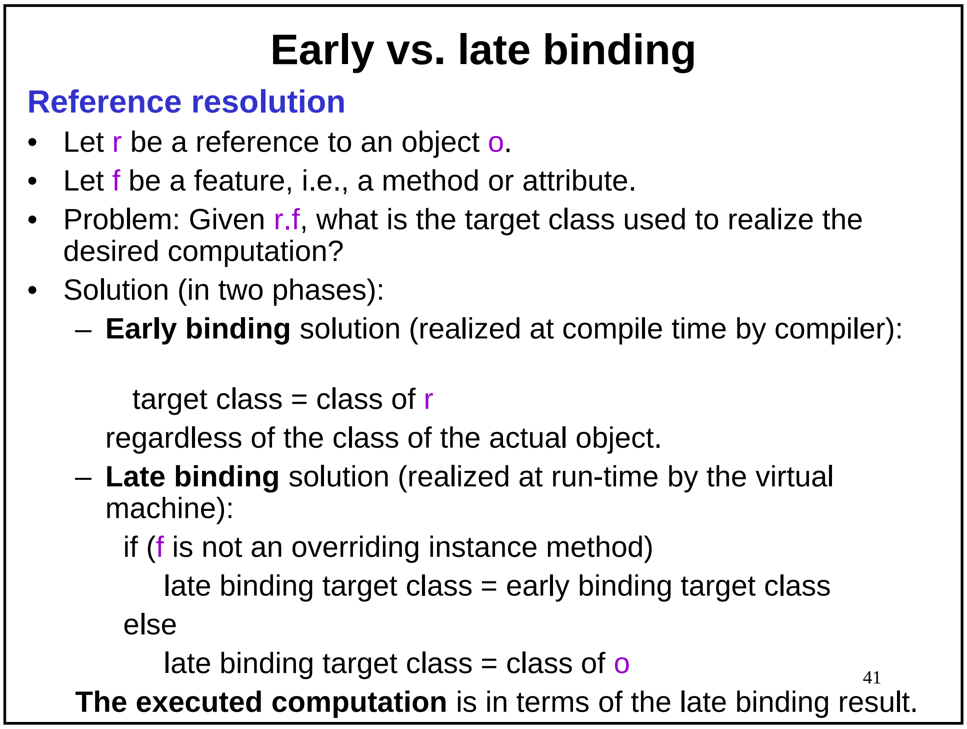
# Inheritance

**Definition and terminology**

- The API of a class C may indicate that it extends some other class P
- Every feature of P is in C
- C inherits from P.
- Child-Parent, Subclass-Superclass
- Inheritance = is-a = Specialization
- Inheritance establishes a chain or hierarchy



UML representation



## Early vs. late binding

### Reference resolution

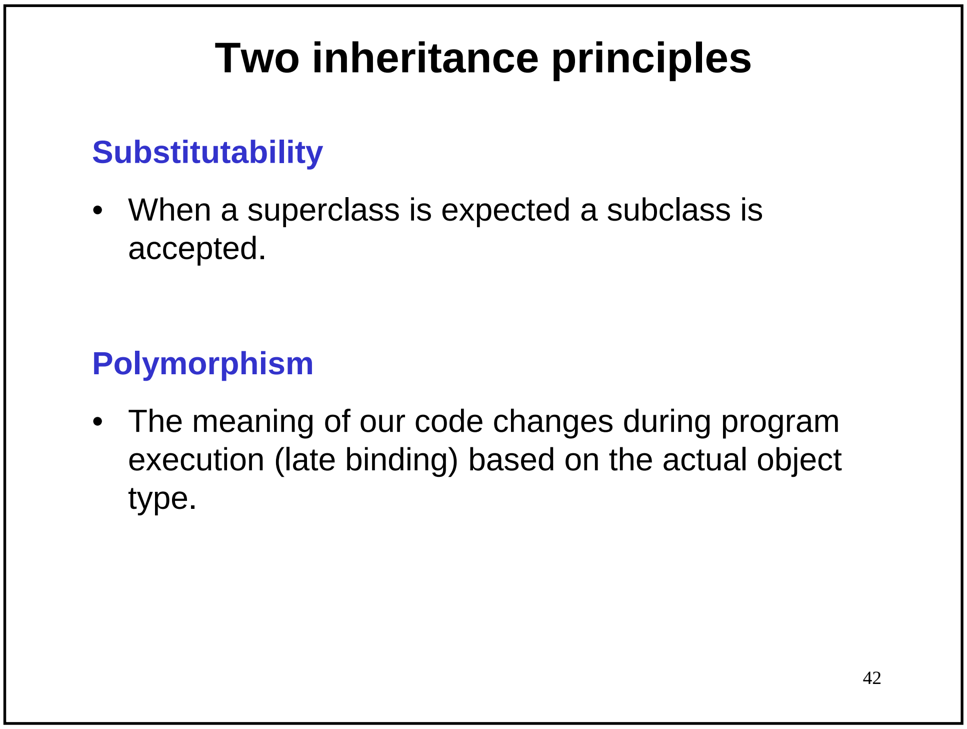
- Let r be a reference to an object o.
- Let f be a feature, i.e., a method or attribute.
- Problem: Given r.f, what is the target class used to realize the desired computation?
- Solution (in two phases):
  - **Early binding** solution (realized at compile time by compiler):

    ```
     target class = class of r
    ```

    regardless of the class of the actual object.
  - **Late binding** solution (realized at run-time by the virtual machine):

    ```
    if (f is not an overriding instance method)
        late binding target class = early binding target class
    else
        late binding target class = class of o
    ```

**The executed computation** is in terms of the late binding result.



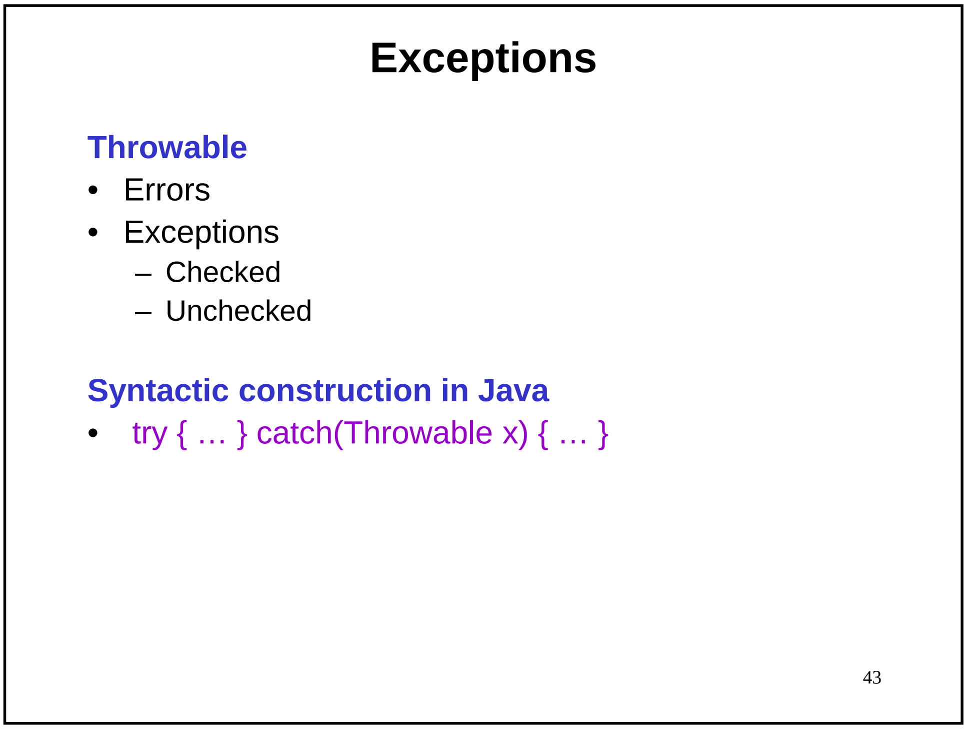
## Two inheritance principles

### Substitutability

- When a superclass is expected a subclass is accepted.

### Polymorphism

- The meaning of our code changes during program execution (late binding) based on the actual object type.



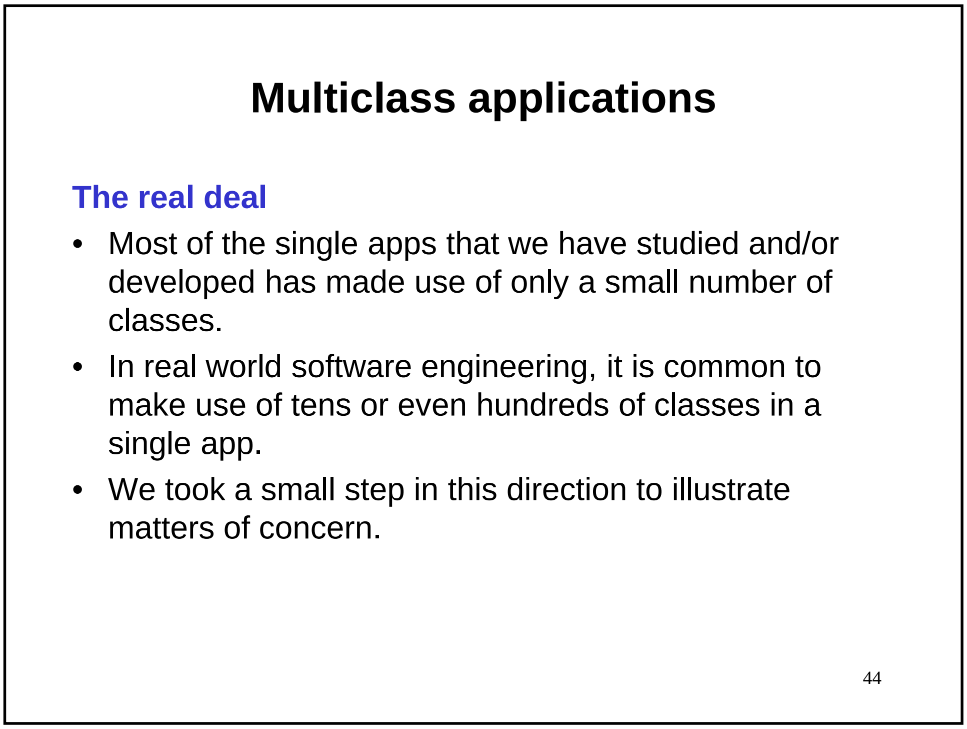
# Exceptions

**Throwable**

- Errors
- Exceptions
  - Checked
  - Unchecked

**Syntactic construction in Java**

- try { … } catch(Throwable x) { … }



# Multiclass applications

**The real deal**

- Most of the single apps that we have studied and/or developed has made use of only a small number of classes.
- In real world software engineering, it is common to make use of tens or even hundreds of classes in a single app.
- We took a small step in this direction to illustrate matters of concern.

## What’s next

**CSE 1030**

- Introduction to Computer Science II
  - Simple data structures
  - Write your own classes

**CSE 2011**

- Introduction to Data Structures

**CSE 3101**

- Design and Analysis of Algorithms

**CSE 3111**

- Software Design

## What’s next

**CSE 1030**

- Introduction to Computer Science II
  - Simple data structures
  - Write your own classes

**CSE 2011**

- Introduction to Data Structures

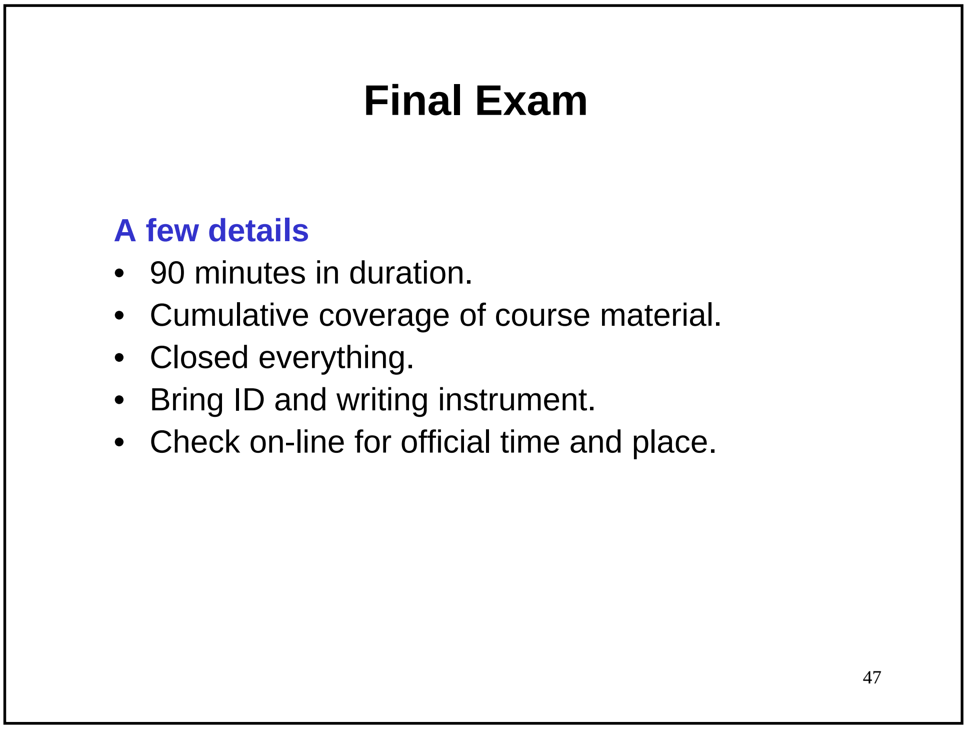
**CSE 3101**

- Design and Analysis of Algorithms

**CSE 3111**

- Software Design

**Many other things as well.**
**But, before any of that ….**
**… the 1020 Final Exam.**



## Final Exam

**A few details**

- 90 minutes in duration.
- Cumulative coverage of course material.
- Closed everything.
- Bring ID and writing instrument.
- Check on-line for official time and place.