CSE 1020: Unit 2

Topic: Delegation, Application Development & Software Engineering

To do: Textbook Chapter 2; Lab 2

1

Outline

- **Delegation**
- **Application development**
- **Software engineering**
- **Software engineering example**

2

Outline

- **Delegation**
- **Application development**
- **Software engineering**
- **Software engineering example**

3

Delegation

Why

- Consider the code inside the box for computing area of a circle.

```
import java.lang.System

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

- It handles both storage (of data) and computation (of area).
- This approach works here, because the there are few variables and the computation is straightforward.

4

Delegation

Why

- Consider the code inside the box for computing area of a circle.

```
import java.lang.System

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

- It handles both storage (of data) and computation (of area).
- This approach works here, because the there are few variables and the computation is straightforward.
- Ultimately, however, we want to build (much) larger software systems and the complexity of the programs would grow too rapidly, if all variables and computation were uniformly exposed.

5

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

6

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

7

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.

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!)

9

Interlude: UML

Unified Modeling Language (UML)

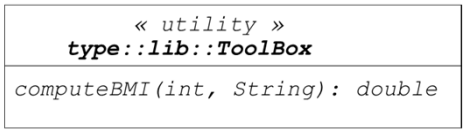
- UML is a visual specification language that allows us to document software in a visual fashion.
- We will consider it in some detail in Unit 7, which is devoted to more advanced concepts in software engineering.
- Here, we introduce its depiction of classes and objects to clarify delegation.

10

Delegation

UML class diagram

- A class diagram depicts critical features of a class that are needed to make use of it.
- Example: The class diagram of a utility in the Type library.

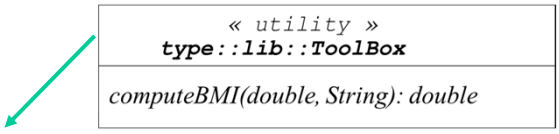


11

Delegation

UML class diagram

- A class diagram depicts critical features of a class that are needed to make use of it.
- Example: The class diagram of a utility in the Type library.



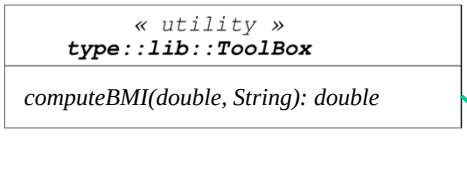
The top box contains the name of the class. Optionally, it is fully qualified (type::lib::ToolBox) and stereotyped (<<utility>>).

12

Delegation

UML class diagram

- A class diagram depicts critical features of a class that are needed to make use of it.
- Example: The class diagram of a utility in the Type library.



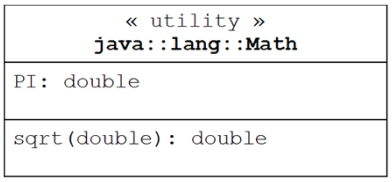
The bottom box contains a list of methods in the class. The list provides the signature of each method: Its name (computeBMI) together with the types of its parameters (double, String). The return type (double) Also is specfied.

13

Delegation

UML class diagram

- A class diagram depicts critical features of a class that are needed to make use of it.
- Example: The class diagram of a utility in the Java library.



Sometimes there is another box, that specifies attributes, here a constant (PI).

14

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

15

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.

16

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?

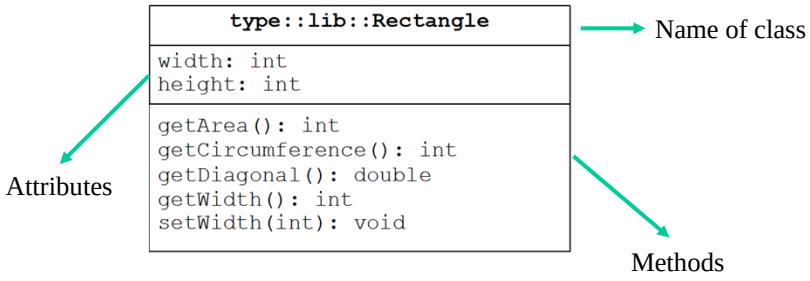
- An object is a software entity that can both store data and perform computation.
- We create an instance (a.k.a. object) of a class using new and the class name.
- The instance has a name, e.g., r, known as the object reference.
- Methods are invoked on the instance (not on the class).
- Each object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

17

Delegation

UML class diagram

- A class diagram depicts critical features of a class that are needed to make use of it.
- Example: The class diagram of a non-utility in the Type library.

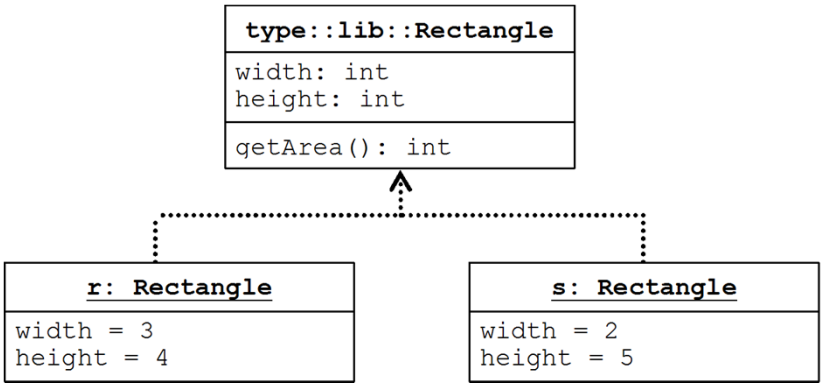


18

Delegation

UML object diagram

- The object diagram is similar to a class diagram, except it focuses on the object's state and identity.
- We indicate how individual objects relate to the class via the instance-of relationship (shown with dotted arrows).



Delegation

A unified view

- When using objects, we copy the class and subsequently use the created copy (i.e., object instance).
  - State is held in the object.
  - We invoke methods on the object.
- Utility classes cannot be copied. So,
  - We invoke their methods on the class.
  - We access their (constant) attributes on the class.
- Either way, we achieve a way to manage complexity by delegating to other resources.

20

Outline

- Delegation
- **Application development**
- Software engineering
- Software engineering example

21

Application development

Applications programming

- Our first program, Hello, is an example of an application (app).
- An app is meant to be run by the Java intepreter to provide a service to an end user.
- An app does not provide anything that can be used by other programmers (only for end users).

Remarks on the internals of an app

- An app is a class that contains one and only one main method.
- The main method runs first when the app starts executing.
- An app may use methods from predefined classes (i.e., it might delegate).

22

Application development

Client vs. Implementer

- The client is the developer of the main class. He understands the big picture, the purpose of the application.
- The implementer is the developer of a component. He focuses on the inner details of the component.
- Separation of concerns means the client and implementer share information on a need-to-know basis.

23

Application development

Client vs. Implementer

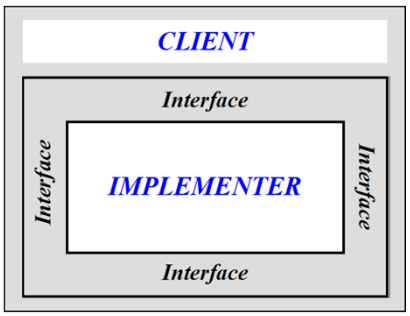
- The client is the developer of the main class. He understands the big picture, the purpose of the application.
- The implementer is the developer of a component. He focuses on the inner details of the component.
- Separation of concerns means the client and implementer share information on a need-to-know basis.
- CSE 1020 focuses on the client; CSE 1030 is more implementer focused.

24

Application development

The client view

- The client knows how to shop for components and how to read their specifications.
- Given a component, the client does not care how it accomplishes its tasks, only what it does.
- The client views a component via its Application Programming Interface (API).
- The class of a component thus encapsulates it.

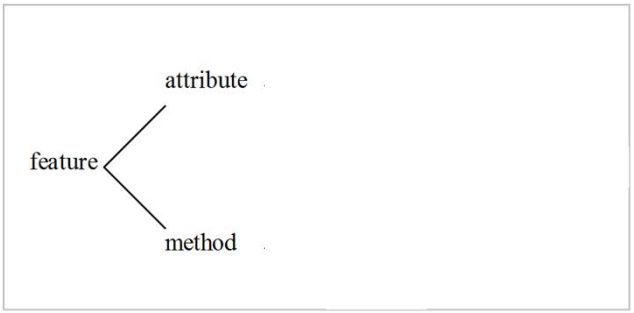


25

Application development

The client view

- A class is made up of features. A feature is an attribute or a method.

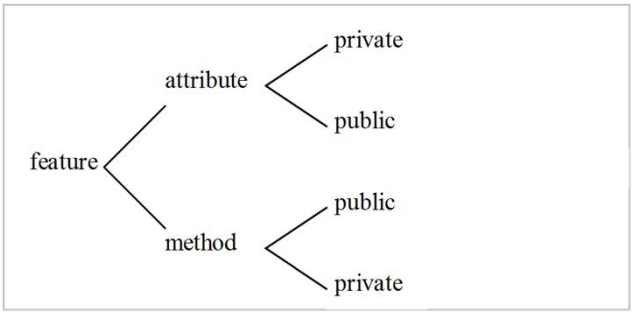


26

Application development

The client view

- A class is made up of features. A feature is an attribute or a method.
- The class of a component classifies each feature as either public or private depending, respectively, on whether the client needs or does not need to know about it.

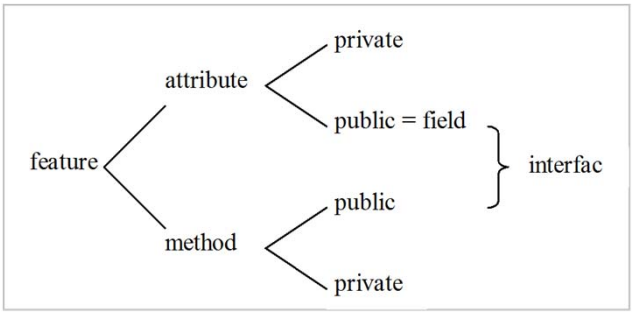


27

Application development

The client view

- A class is made up of features. A feature is an attribute or a method.
- The class of a component classifies each feature as either public or private depending, respectively, on whether the client needs or does not need to know about it.
- The API (interface) of a component lists only the headers of its public methods and the declarations of its public attributes (a.k.a. fields).



28

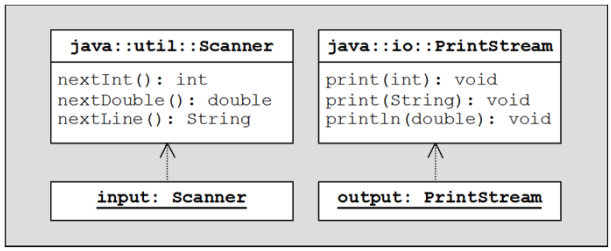
Ready-made I/O components

Keyboard input

```
Scanner input = new Scanner(System.in);
int width = input.nextInt();
```

Screen output

```
PrintStream output = System.out;
output.print(width);
```



29

Outline

- Delegation
- Application development
- **Software engineering**
- Software engineering example

30

Software engineering

A methodology for developing software

- Development of software is similar to development in other areas of engineering.
- We seek to systematically apply scientific knowledge to the solution of practical software problems.

31

Software engineering

Contracts

- Each method in a component comes with a contract that spells out the responsibilities of the client and the implementer.
- The client must supply parameters that satisfy the precondition of the method.
- The implementer must supply a return that satisfy the postcondition of the method.
- Liability:
  - If pre=false, the client is at fault.
  - If pre=true and post=false then the implementer is at fault.
  - If pre=post=true then everything is OK.
- Remark: if a method has pre=true then its client does not have to ensure anything.

32

Software engineering

Contracts

- Methods in the Java standard library specify their pre and post as follows:
  - pre is always true unless stated otherwise
  - post is specified under Returns and Throws
- Example: This contract specifies pre=true (i.e. no condition on the parameter). The post states that the method will return the square root if x is non-negative and will throw an exception otherwise.

`double squareRoot(double x)`
Returns the square root of the given argument.

Parameters:
`x` – an argument.
Returns:
the positive square root of `x`.
Throws:
an exception if `x` < 0.

33

Software engineering

Two guidelines

1. Risk mitigation by early exposure: If you are not sure about something during software development, confront it as early as possible. Making changes later is more difficulty than doing so now.

   Example: the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error. The risk of truncating the real value is exposed early.

Software engineering

Two guidelines

1. Risk mitigation by early exposure: If you are not sure about something during software development, confront it as early as possible. Making changes later is more difficulty than doing so now.

   Example: the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error. The risk of truncating the real value is exposed early.

2. Handling constants: Replace all magic numbers (literals) in your program with finals.

   Example:

   Instead of:

   ```
   width = width / 12;
   ```

   Write:

   ```
   final int INCH_PER_FOOT = 12;
   width = width / INCH_PER_FOOT;
   ```

Software engineering

Two guidelines

1. Risk mitigation by early exposure: If you are not sure about something during software development, confront it as early as possible. Making changes later is more difficulty than doing so now.

   Example: the Java compiler turns a potential logic error (like assigning a real value to an int variable) to a compile-time error. The risk of truncating the real value is exposed early.

2. Handling constants: Replace all magic numbers (literals) in your program with finals.

   Example:

   Instead of:

   ```
   width = width / 12;
   ```

   Write:

   ```
   final int INCH_PER_FOOT = 12;
   width = width / INCH_PER_FOOT;
   ```

Compared to the above:

- The name of the constant is self documenting.
- Specification as final allows compiler to prevent you from inadvertently changing the value.

Software engineering

Phased development

1. Requirements

Software engineering

Phased development

1. Requirements
    1.1 Problem definition

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.

40

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)

41

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.

42

Software engineering

Phased development

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.

43

Software engineering

Phased development

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

44

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

Remark

- This classical paradigm now augmented to include early prototyping for user feedback.

45

Software engineering

Phased development

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

Remark

Waterfall method

- This classical paradigm now augmented to include early prototyping for user feedback.

46

Software engineering

Phased development

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

Remark

Waterfall method

- This classical paradigm now augmented to include early prototyping for user feedback.

Spiral development

47

Software engineering

Phased development

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

Remark

- This classical paradigm now augmented to include early prototyping for user feedback.

48

Software engineering

3 steps in implementation

1. Edit
2. Compile
3. Run

49

Software engineering

Edit

- In an editor we enter

Remember this?

```
import type.lang.*;

public class Hello
{    public static void main(String[ ] args)
     {     IO.println("Hello, world!");
     }
}
```

- and save to a file Hello.java

50

Software engineering

3 steps in implementation

1. Edit
2. Compile
3. Run

51

Software engineering

Compile

- At our command line prompt…
- we invoke the compiler…
- to produce byte code to be interpreted by computer.

% javac Hello.java

52

Software engineering

Typically initial attempts to compile

- Yield errors.
- These compile-time errors are usually errors of syntax in your programming (sometimes called syntax errors).
- The compiler will produce diagnostic messages.

53

Software engineering

Typically initial attempts to compile

- Yield errors.
- These compile-time errors are usually errors of syntax in your programming (sometimes called syntax errors).
- The compiler will produce diagnostic messages.

Response

- Return to editor.
- Correct errors.
- Reattempt compile.
- Repeat until no more error messages…
- … and the compiler has produced Hello.class
  - The byte code version that can be interpreted by computer (via the Java interpreter).

54

Software engineering

3 steps in implementation

1. Edit
2. Compile
3. Run

55

Software engineering

Run

- We now convert the byte code produced by the compiler…
- … to native code that executes on the machine at hand.
- At the command line prompt we invoke the interpreter

% java Hello

Success

- Will produce on the screen

Hello, world!

56

Software engineering

Run

- We now convert the byte code produced by the compiler…
- … to native code that executes on the machine at hand.
- At the command line prompt we invoke the interpreter

% java Hello

Failure

1. Run-time errors/crashes → attempt syntactically correct; but, illegal operation.
   - Return to editor and iterate process until correct.

57

Software engineering

Run

- We now convert the byte code produced by the compiler…
- … to native code that executes on the machine at hand.
- At the command line prompt we invoke the interpreter

% java Hello

Failure

2. Logical errors → Program syntactically okay, executes; but, produces incorrect output.
   - May require return to design, analysis or definition.

58

Software engineering

Run

- We now convert the byte code produced by the compiler…
- … to native code that executes on the machine at hand.
- At the command line prompt we invoke the interpreter

% java Hello

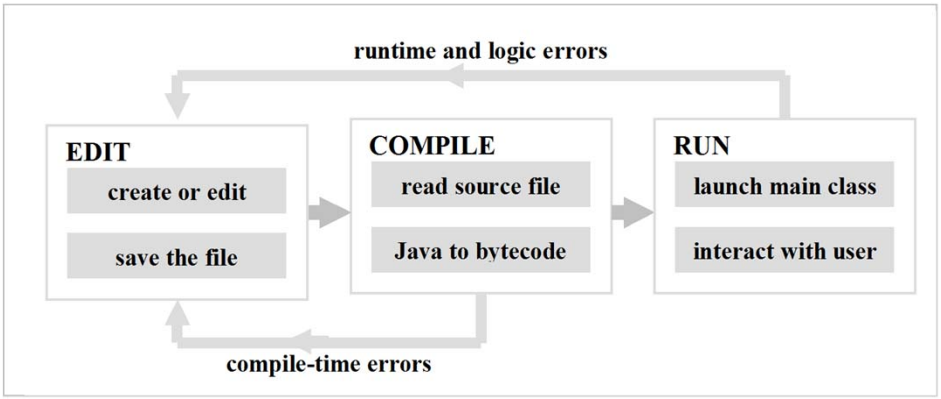
Success

- Will produce on the screen

Hello, world!

59

Software engineering



60

Software engineering

General program structure

- Declaration
- Input
- Computation
- Output

61

Software engineering

DICO analysis example

```
/*
Java program to print a greeting. Upon invocation it
prints “Hello, world!” to standard out.

Author: Richard Wildes                    Date: 05/14/13
*/

Import type.lang.*; // import type package for general utils.

// Definition of the Hello class.
public class Hello
{   public static void main(String[ ] args)
    {   // Print to standard out.
        IO.println(“Hello, world”);
    }
}
```

D

O

62

Software engineering

DICO analysis example

```
/*
Java program to print a greeting. Upon invocation it
prints “Hello, world!” to standard out.

Author: Richard Wildes                    Date: 05/14/13
*/

import type.lang.*; // import type package for general utils.

// Definition of the Hello class.
public class Hello
{   public static void main(String[ ] args)
    {    // Print to standard out.
         IO.println(“Hello, world”);
    }
}
```

D

O

Remark: I and C are vacuous for this simple program.

63

A template for CSE 1020 programs

```
/*
Class to ???.

Author: ???                               Date: ???
*/

import type.lang.*;
import java.util.Scanner;
import java.io.PrintStream;

// Definition of the ClassName class.
public class ClassName
{   public static void main(String[ ] args)
    {   Scanner input = new Scanner(System.in);
        PrintStream output = System.out;
        // App specific DICO.
    }
}
```

64

A template for CSE 1020 programs

```
/*
Class to ???.

Author: ???                                   Date: ???
*/

import type.lang.*;
import java.util.Scanner;
import java.io.PrintStream;

// Definition of the ClassName class.
public class ClassName
{   public static void main(String[ ] args)
    {   Scanner input = new Scanner(System.in);
        PrintStream output = System.out;
        // App specific DICO.
    }
}
```

Preferred 1020 I/O. Recall: We introduced these earlier in this unit.

65

Software engineering

First pass has introduced

- Contracts
- A few guidelines: Risk mitigation; dealing with constants
- Phased development.
- A look at error types.
- Abstraction to basic program form: DICO.
- Program template.

Remark

- For a nice discussion of programming style in Java see Textbook Appendix C.

66

Outline

- Delegation
- Application development
- Software engineering
- **Software engineering example**

67

Software engineering example

Phased development: A program to make change

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

68

Software engineering example

Phased development: A program to make change

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

69

Software engineering example

Requirements: Problem definition

- Want a program that can calculate the conversion of an amount of money (CND) into a corresponding amount of change (quarters, dimes, nickels and pennies).

70

Software engineering example

Requirements: Analysis

- **Input:** Amount of money in cents.
- **Output:** The corresponding amount of change (quarters, dimes, nickels and pennies).
- **Format:**
  Change is q quarters, d dimes, n nickels, p pennies.

71

Software engineering example

Phased development: A program to make change

1. Requirements
   1.1 Problem definition → general description.
   1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

72

Software engineering example

Design

- **Algorithm:** Let's do it the way people do it.

67 cents
| --------→ 2 quarters
17 cents
| --------→ 1 dime
7 cents
| --------→ 1 nickel
2 cents
| --------→ 2 pennies

73

Software engineering example

Design

- **Algorithm:** Let's do it the way people do it.

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.
4. Repeat steps 1 & 2 for nickels.
5. The final remainder is the number of pennies.

74

Software engineering example

Design

- **Algorithm:** Let's do it the way people do it.

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.
4. Repeat steps 1 & 2 for nickels.
5. The final remainder is the number of pennies.

- **Variables:** In red are likely variables or constants

75

Software engineering example

Design

- **Algorithm:** Let's do it the way people do it.

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.
4. Repeat steps 1 & 2 for nickels.
5. The final remainder is the number of pennies.

- **Variables:** In red are likely variables or constants
  - Type int is appropriate as algorithm uses integer operations.

76

Software engineering example

Phased development: A program to make change

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

77

Software engineering example

Implementation: General program structure

- Declaration
- Input
- Computation
- Output

78

Software engineering example

Our Template

```
/*
Class to ???.

Author: ???                                   Date: ???
*/

import type.lang.*;
import java.util.Scanner;
import java.io.PrintStream;

// Definition of the ClassName class.
public class ClassName
{   public static void main(String[ ] args)
    {   Scanner input = new Scanner(System.in);
        PrintStream output = System.out;
        // App specific DICO.
    }
}
```

79

Software engineering example

Make change skeleton

```
/*
Class to make change. Based on an example by YL.

Author: Richard Wildes                        Date: 05/16/2013
*/

import type.lang.*;
import java.util.Scanner;
import java.io.PrintStream;

// Definition of the MkChange class.
public class MkChange
{   public static void main(String[ ] args)
    {   Scanner input = new Scanner(System.in);
        PrintStream output = System.out;
        // App specific DICO.
    }
}
```

80

Software engineering example

Fill in the DICO details

```
// Definition of the MkChange class.
public class MkChange
{   public static void main(String[ ] args)
    {
        // Declaration

        // Input

        // Computation

        // Output
    }
}
```

81

Software engineering example

DICO: Declaration

```
// Declaration.
```

82

Software engineering example

DICO: Declaration

```
// Declaration.
Scanner input = new Scanner(System.in);
PrintStream output = System.out;
```

83

Software engineering example

DICO: Declaration

// Declaration.

- During design we had isolated that we would want to manipulate data having to do with
  - Amount of input money
  - Value of quarter, dimes and nickels
  - Number of quarters, dimes, nickels and pennies

84

Software engineering example

DICO: Declaration

// Declaration.

- During design we had isolated that we would want to manipulate data having to do with
  - Amount of input money
  - Value of quarter, dimes and nickels
  - Number of quarters, dimes, nickels and pennies
- Some of these entities will not change during our computation: These should be constants.

85

Software engineering example

DICO: Declaration

```
// Declaration.
final int QUARTER_VALUE = 25;
final int DIME_VALUE = 10;
final int NICKEL_VALUE = 5;
```

86

Software engineering example

DICO: Declaration

// Declaration

- During design we had isolated that we would want to manipulate data having to do with
  - Amount of input money
  - Value of quarter, dimes and nickels
  - Number of quarters, dimes, nickels and pennies
- Some of these entities will not change during our computation: These should be constants.
- Some will vary during the computation: Variables.

87

Software engineering example

DICO: Declaration

```
// Declaration
final int QUARTER_VALUE = 25;
final int DIME_VALUE = 10;
final int NICKEL_VALUE = 5;
int amount, nQuarters, nDimes, nNickels, nPennies;
```

88

Software engineering example

DICO: Input

// Input.

Software engineering example

DICO: Input

// Input.

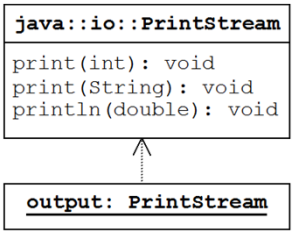
- We need to prompt user for an input amount,

Software engineering example

DICO: Input

// Input.

- We need to prompt user for an input amount,



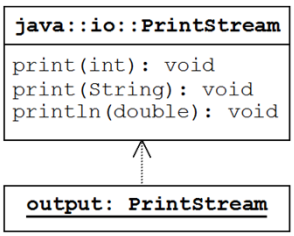
91

Software engineering example

DICO: Input

// Input.

- We need to prompt user for an input amount,
  - In the PrintStream class we find print



92

Software engineering example

DICO: Input

// Input.

- We need to prompt user for an input amount,
  - In the PrintStream class we find print
- We need to read the user's response (an int)

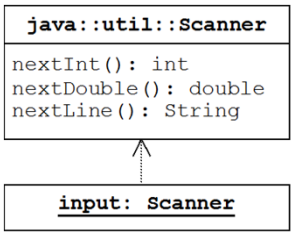
93

Software engineering example

DICO: Input

// Input.

- We need to prompt user for an input amount,
  - In the PrintStream class we find print
- We need to read the user's response (an int)



94

Software engineering example

DICO: Input

// Input.

- We need to prompt user for an input amount,
  - In the PrintStream class we find print
- We need to read the user's response (an int)
  - In the Scanner class we find nextInt

95

Software engineering example

DICO: Input

```
// Input.
output.print("Enter the amount in cents: ");
amount = input.nextInt();
```

96

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors in incrementally written code is far easier than working with larger chunks.

97

Software engineering example

DICO: Input

```
// Input
output.print("Enter the amount in cents: ");
amount = input.nextInt();
```

98

Software engineering example

DICO: Input

```
// Input
output.print(“Enter the amount in cents: ”);
amount = input.nextInt();
// following are 2 test/debugging statements
output.print(“After input amount is: ”);
output.println(amount);
```

99

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

%

100

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
```

101

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
%
```

102

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
```

103

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents:
```

104

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 100
```

105

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 100
After input amount is: 100
%
```

106

Software engineering example

DICO: Input

```
// Input
output.print(“Enter the amount in cents: ”);
amount = input.nextInt();
// following are 2 test/debugging statements
output.print(“After input amount is:”);
output.println(amount);
```

107

Software engineering example

DICO: Input

```
// Input
output.print(“Enter the amount in cents: ”);
amount = input.nextInt();
/* following are 2 test/debugging statements
output.print(“After input amount is:”);
output.println(amount);
*/
```

108

Software engineering example

DICO: Computation

// Computation

Software engineering example

DICO: Computation

// Computation

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
```

111

Software engineering example

DICO: Computation

```
// Computation
```

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.

112

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
```

113

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors in incrementally written code is far easier than working with larger chunks.

114

Software engineering example

DICO: Computation

```
// Computation.
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
```

115

Software engineering example

DICO: Computation

```
// Computation.
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
// following are 4 test/debugging statements
output.print("After calculation and removal of quarters amount is ");
output.print(amount);
output.print(" and quarters are ");
output.println(nQuarters);
```

116

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
%
```

117

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
```

118

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
%
```

119

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
```

120

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents:
```

121

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 110
```

122

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 110
After calculation and removal of quarters amount
is 10 and quarters are 4
```

123

Software engineering example

DICO: Computation

```
// Computation.
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
// following are 4 test/debugging statements
output.print("After calculation and removal of quarters amount is ");
output.print(amount);
output.print(" and quarters are ");
output.println(nQuarters);
```

124

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
/* following are 4 test/debugging statements
output.print("After calculation and removal of quarters amount is ");
output.print(amount);
output.print(" and quarters are ");
output.println(nQuarters);
*/
```

125

Software engineering example

DICO: Computation

// Computation

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.

126

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
```

127

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
```

128

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
```

129

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors in incrementally written code is far easier than working with larger chunks.

130

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
// following are 6 test/debugging statements
output.print("After calculation and removal of dimes amount is ");
output.print(amount);
output.print(" and quarters are ");
output.print(nQuarters);
output.print(" and dimes are ");
output.println(nDimes);
```

131

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

%

132

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
```

133

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
%
```

134

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
```

135

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents:
```

136

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 113
```

137

Software engineering example

Incremental edit/compile/run

- About now would be a good time to test what has been done so far.
- Finding errors incrementally written code is far easier than working with larger chunks.

```
% javac MkChange.java
% java MkChange
Enter the amount in cents: 113
After calculation and removal of dimes amount
is 3 and quarters are 4 and dimes are 1
```

138

Software engineering example

DICO: Computation

```
// Computation.
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
// following are 6 test/debugging statements
output.print("After calculation and removal of dimes amount is ");
output.print(amount);
output.print(" and quarters are ");
output.print(nQuarters);
output.print(" and dimes are ");
output.println(nDimes);
```

139

Software engineering example

DICO: Computation

```
// Computation.
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
/* following are 6 test/debugging statements
output.print("After calculation and removal of dimes amount is ");
output.print(amount);
output.print(" and quarters are ");
output.print(nQuarters);
output.print(" and dimes are ");
output.println(nDimes);
*/
```

140

Software engineering example

DICO: Computation

// Computation

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.
4. Repeat steps 1 & 2 for nickels.

141

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
nNickels = amount / NICKEL_VALUE;
amount = amount % NICKEL_VALUE;
```

142

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
nNickels = amount / NICKEL_VALUE;
amount = amount % NICKEL_VALUE;
```

Remark

- Now is a good time for incremental edit/compile/run.

143

Software engineering example

DICO: Computation

// Computation

1. Calculate the maximum quarters that you can use.
   - Divide the amount by the quarter value.
   - The integer part of the result is the number of quarters
2. Remove the quarters from the amount
   - Set amount to the remainder of the previous division.
3. Repeat steps 1 & 2 for dimes.
4. Repeat steps 1 & 2 for nickels.
5. The final remainder is the number of pennies.

144

Software engineering example

DICO: Computation

```
// Computation
nQuarters = amount / QUARTER_VALUE;
amount = amount % QUARTER_VALUE;
nDimes = amount / DIME_VALUE;
amount = amount % DIME_VALUE;
nNickels = amount / NICKEL_VALUE;
amount = amount % NICKEL_VALUE;
nPennies = amount;
```

Remark

- Now is a good time for incremental edit/compile/run.

145

Software engineering example

DICO: Output

```
// Output
```

146

Software engineering example

DICO: Output

// Output.

- **Format:**
  Change is q quarters, d dimes, n nickels, p pennies.

147

Software engineering example

DICO: Output

```
// Output.
output.print(“Change is ”);
```

148

Software engineering example

DICO: Output

```
// Output.
output.print(“Change is ”);
output.print(nQuarters + “ quarters, ”);
```

149

Software engineering example

DICO: Output

```
// Output.
output.print(“Change is ”);
output.print(nQuarters + “ quarters, ”);
output.print(nDimes + “ dimes, ”);
```

150

Software engineering example

DICO: Output

```
// Output
output.print(“Change is ”);
output.print(nQuarters + “ quarters, ”);
output.print(nDimes + “ dimes, ”);
output.print(nNickels + “ nickels, ”);
```

151

Software engineering example

DICO: Output

```
// Output
output.print(“Change is ”);
output.print(nQuarters + “ quarters, ”);
output.print(nDimes + “ dimes, ”);
output.print(nNickels + “ nickels, ”);
output.println(nPennies + “ pennies.”);
```

152

Software engineering example

Completing the implementation cycle

- We now save our code to a file MkChange.java…
- …and continue with the edit/compile/run cycle until
- …we have nominally working MkChange.class

153

Software engineering example

Completing the implementation cycle

- We now save our code to a file MkChange.java…
- …and continue with the edit/compile/run cycle until
- …we have nominally working MkChange.class

Remarks

- We have been working *incrementally* through the edit/compile/run cycle as we wrote our code.
- Therefore, final compilation will go relatively smoothly.
- Such incremental implementation has a significant positive impact on:
  - the quality of code produced;
  - minimizing the time required to yield working programs.

154

Software engineering example

Phased development: A program to make change

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

155

Software engineering example

Test

%

156

Software engineering example

Test

% java MkChange

157

Software engineering example

Test

% java MkChange
Enter the amount in cents:

158

Software engineering example

Test

% java MkChange
Enter the amount in cents: 67

159

Software engineering example

Test

% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
%

160

Software engineering example

Test

% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
% java MkChange

161

Software engineering example

Test

% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
% java MkChange
Enter the amount in cents:

162

Software engineering example

Test

```
% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
% java MkChange
Enter the amount in cents: 0
```

163

Software engineering example

Test

```
% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
% java MkChange
Enter the amount in cents: 0
Change is 0 quarters, 0 dimes, 0 nickels, 0 pennies.
%
```

164

Software engineering example

Test

```
% java MkChange
Enter the amount in cents: 67
Change is 2 quarters, 1 dimes, 1 nickels, 2 pennies.
% java MkChange
Enter the amount in cents: 0
Change is 0 quarters, 0 dimes, 0 nickels, 0 pennies.
%
```

Remark

- In practice, would submit program to a more extensive battery of tests.

165

Software engineering example

Phased development: A program to make change

1. Requirements
   - 1.1 Problem definition → general description.
   - 1.2 Analysis → Input & validation; Output and format.
2. Design → representation and procedures (data structures and algorithms)
3. Implementation → Program.
4. Testing → Empirical evaluation.
5. Deployment (incl. Maintenance) → fielded product.

166

Software engineering example

Deployment

- In real life you now ship/install your product.
- Here, just for fun, I've placed the source code on our section website.

167

Software engineering example

Recapitulation

- We have gone through our first nontrivial software engineering example.
- Followed the phased development methodology.
- Made use of our Java program template.
- Coded the program in terms of DICO structure.
- Employed incremental approach to implementation.
- This exercise has yielded our second app.

168

Summary

- **Delegation**
- **Application development**
- **Software engineering**
- **Software engineering example**

169

Appendix: The Java standard library

Three major components

1. J2SE (Java 2 Standard Edition): For developing desktop applications.
2. J2EE (Java 2 Enterprise Edition): For developing enterprise-wide and sever applications.
3. J2ME (Java 2 Micro Edition): For developing consumer space applications.

- See the maker's website java.sun.com

170

Appendix: The Java standard library

The major components

1. J2SE (Java 2 Standard Edition): For developing desktop applications.
2. J2EE (Java 2 Enterprise Edition): For developing enterprise-wide and sever applications.
3. J2ME (Java 2 Micro Edition): For developing consumer space applications.

- See the maker's website java.sun.com
- Of particular interest to us is J2SE.

171

Appendix: The Java standard library

The J2SE JDK (J2SE Development Kit) 2 components

1. J2SE Runtime Environment: This JRE contains all that is needed to run Java applications, including
   - Class library
   - Virtual machine
2. Tools: Programs needed to develop Java applications, including
   - The Java compilerJ2SE (Java 2 Standard Edition): For developing desktop applications.

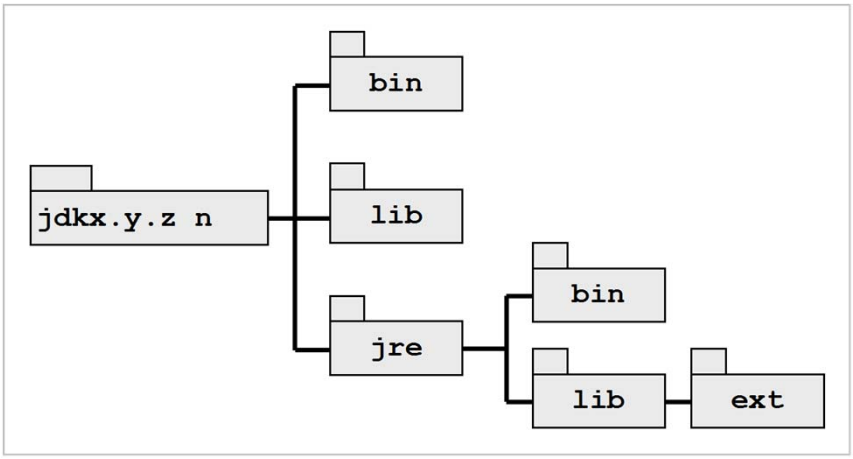
- The JRE alone is not sufficient for developing apps.

172

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

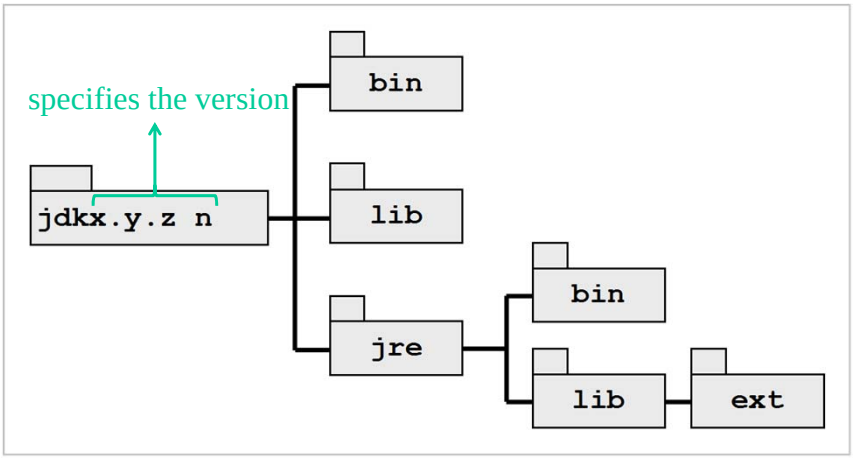


173

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

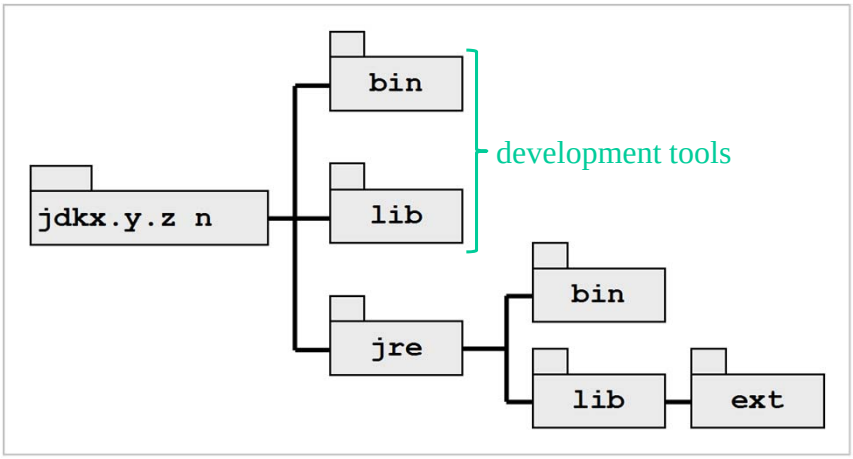


174

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

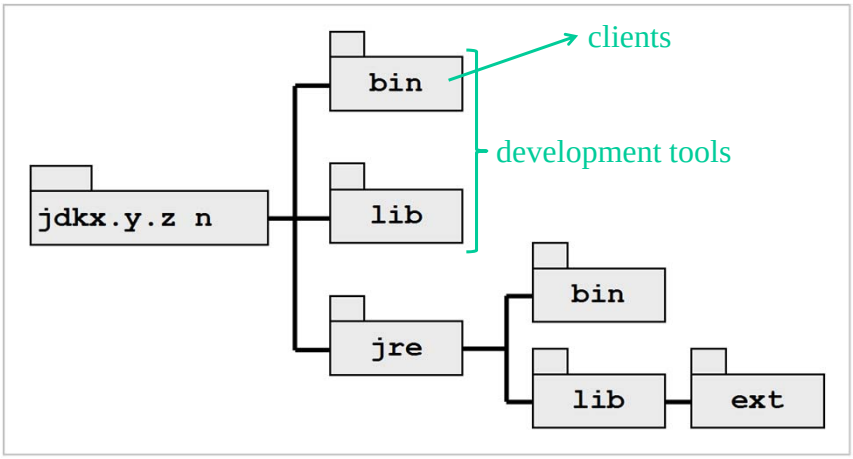


175

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

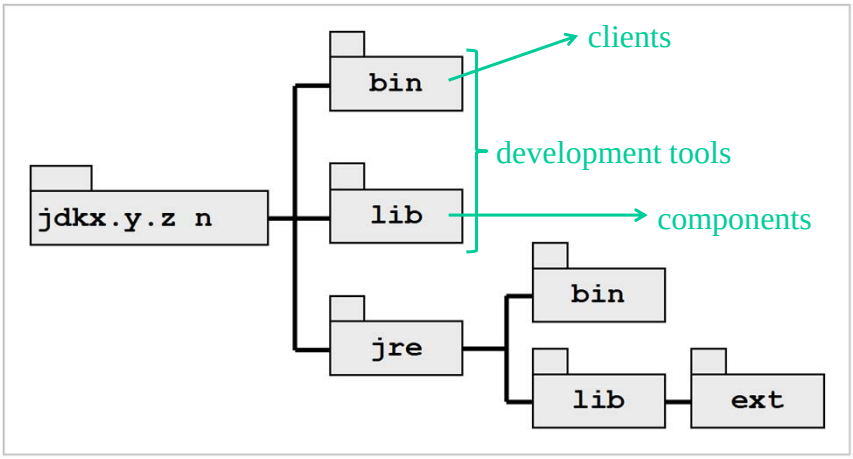


176

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

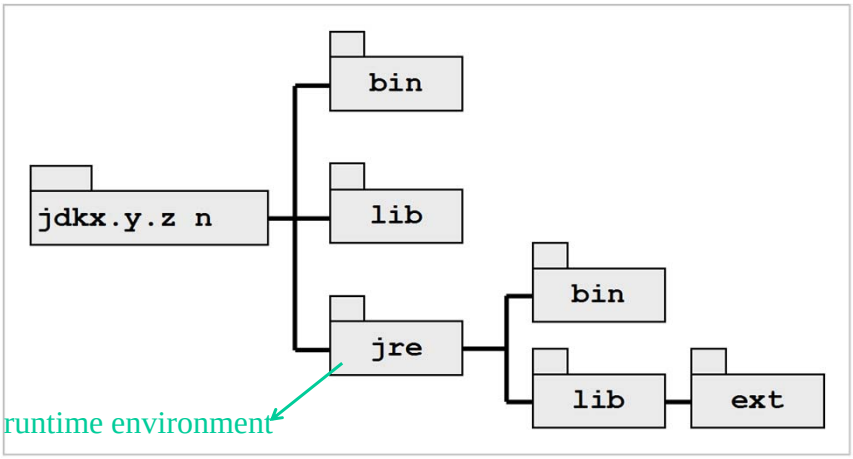


177

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.

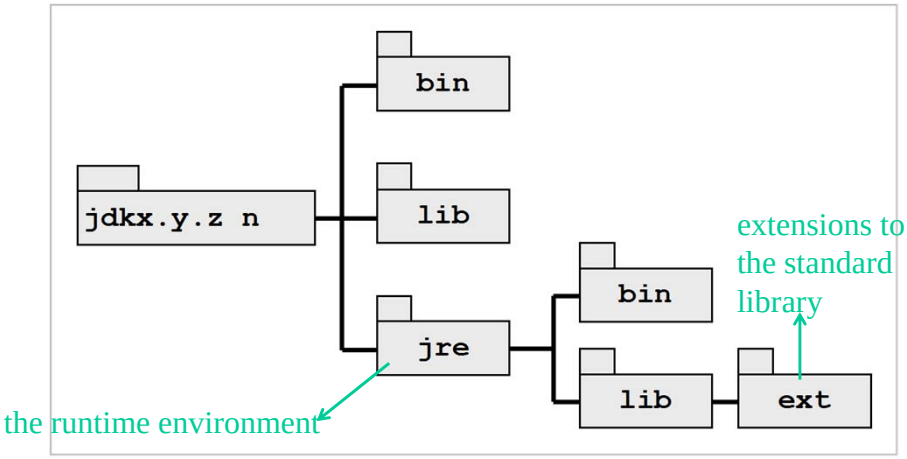


178

Appendix: The Java standard library

Overview of the JDK files

- The JDK will install to directory with the depicted organization.



179

Appendix: The Java standard library

Class library overview

- The classes are organized in packages and subpackages.
- Top level packages are shown on the RHS of this slide.
- More discussion will be forthcoming.

`java.awt`	Provides support for drawing graphics. AWT = Abstract Windowing Toolkit
`java.beans`	Provide support for Java Beans.
`java.io`	Provides support for file and other I/O operations.
`java.lang`	Provides the fundamental Java classes. This package is auto-imported by the compiler.
`java.math`	Provides support for arbitrary-precision arithmetic
`java.net`	Provides support for network access.
`java.rmi`	Provides support for RMI. RMI = Remote Method Invocation
`java.security`	Provides support for the security framework.
`java.sql`	Provides support for databases access over JDBC JDBC = Java Database Connectivity, SQL = Structured Query Language
`java.text`	Provides formatting for text, dates, and numbers.
`java.util`	Miscellaneous utility classes including JCF. JCF = Java Collection Framework
`javax.crypto`	Provides support for cryptographic operations.
`javax.servlet`	Provides support for servlet and JSP development. JSP = Java Server Pages
`javax.swing`	Provides support for GUI development. GUI = Graphical User Interface
`javax.xml`	Provides support for XML processing. XML = eXtensible Markup Language