Computer Programming I – CSC111
Chapter 5 – Defining Classes and Methods

Dr. Mejdl Safran

mejdl@ksu.edu.sa

Outline

- Class Files and Separate Compilation
- Instance Variables
- Methods
- The Keyword this
- Local Variables
- Blocks
- Parameters of a Primitive Type

Class and Method Definitions

- Java program consists of objects
  - Objects of class types
  - Objects that interact with one another
- Program objects can represent
  - Objects in real world
  - Abstractions

Class and Method Definitions

- A class as a blueprint

Class name: Account
Data: • ID • Name • Balance
Methods (actions): • Credit amount • Debit amount • Display info

Class and Method Definitions

- A class outline as a UML class diagram

Account
+ ID: String + name: String + balance: double
+ credit (amount): double + debit (amount): double + display (): void

Class and Method Definitions

Object name: acc1

ID: 1111
Name: Mohammad
Balance: 2000

Object name: acc2

ID: 2222
Name: Saad
Balance: 2500

Object name: acc3

ID: 3333
Name: Fahad
Balance: 2000

Objects that are instantiations of the class `Account`

Bank account class and instance variables

```
public class Account {
    public String id;
    public String name;
    public double balance;

    public void display(){
        System.out.println("\tAccount information");
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Balance: " + balance);
        System.out.println();
    }
    public double balanceInDollars() {
        double balanceDollars;
        balanceDollars = balance / 3.75;
        return balanceDollars;
    }
}
```

- Note class has
  - Three pieces of data (instance variables)
  - Two behaviors
- Each instance of this type has its own copies of the data items
- Use of **public**
  - No restrictions on how variables used
  - Later will replace with **private**

Using a class and its methods

```
public class AccountTest {
    public static void main(String[] args) {

    Account acc1 = new Account();
    acc1.id = "1111";
    acc1.name = "Mohammad";
    acc1.balance = 3000;
    acc1.display();
    Account acc2 = new Account();
    acc2.id = "2222";
    acc2.name ="Saad";
    acc2.balance = 1000;
    acc2.display();

    double balanceDollars;
    balanceDollars = acc1.balanceInDollars();
    System.out.println("Balance of " + acc1.name
    + " in dollars is "+ balanceDollars);
    }
}
```

```
Account information
ID: 1111
Name: Mohammad
Balance: 3000.0

Account information
ID: 2222
Name: Saad
Balance: 1000.0

Balance of Mohammad in dollars is 800.0
```

Sample screen output

Class Files and Separate Compilation

- Each `Java` class definition usually in a file by itself
  - File begins with name of the class
  - Ends with `.java`
- Class can be compiled separately
- Helpful to keep all class files used by a program in the same directory

Methods

- When you use a method you "invoke" or "call" it
- Two kinds of Java methods
  - Return a single item
  - Perform some other action – a `void` method
- The method `main` is a `void` method
  - Invoked by the system
  - Not by the application program

Methods

- Calling a method that returns a quantity
  - Use anywhere a value can be used
  - e.g., `double` y = acc1. `balanceInDollars();`
- Calling a void method
  - Write the invocation followed by a semicolon
  - Resulting statement performs the action defined by the method
  - E.g., acc1.`display();`

Defining `void` Methods

- Consider method `display`

```
public void display(){
    System.out.println("\tAccount information");
    System.out.println("ID: " + id);
    System.out.println("Name: " + name);
    System.out.println("Balance: " + balance);
    System.out.println();
}
```

- Method definitions appear inside class definition
  - Can be used only with objects of that class

Defining `void` Methods

- Most method definitions we will see as `public`
- Method does not return a value
  - Specified as a `void` method
- Heading includes parameters
- Body enclosed in braces `{ }`
- Think of method as defining an action to be taken

Methods That Return a Value

- Consider method **`balanceInDollars()`**

```
public double balanceInDollars() {
    double balanceDollars;
    balanceDollars = balance / 3.75;
    return balanceDollars;
}
```

- Heading declares type of value to be returned
- Last statement executed is **`return`**

Extend Bank account class: read input

```
public class Account {
    public String id;
    public String name;
    public double balance;

    public void readInput(){
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the account number: ");
        id = keyboard.nextLine();
        System.out.println("Enter the account holder name: ");
        name = keyboard.nextLine();
        System.out.println("Enter the account balance in riyals: ");
        balance = keyboard.nextDouble();
    }
    // the rest of the previously defined methods
}
```

Extend Bank account class: read input

```
public class AccountTest {
    public static void main(String[] args) {

        Account acc1 = new Account();
        acc1.readInput();
        acc1.display();
        Account acc2 = new Account();
        acc2.readInput();
        acc2.display();
    }
}
```

The Keyword `this`

- Referring to instance variables outside the class – must use
  - Name of an object of the class
  - Followed by a dot
  - Name of instance variable
- Inside the class,
  - Use name of variable alone
  - The object (unnamed) is understood to be there

The Keyword `this`

- Inside the class the unnamed object can be referred to with the name `this`
- Example

```
this.name = keyboard.nextLine();
```

- The keyword `this` stands for the receiving object
- We will see some situations later that require the `this`

Local Variables

- Variables declared inside a method are called *local* variables
  - May be used only inside the method
  - All variables declared in method `main` are local to `main`
- Local variables having the same name and declared in different methods are different variables

Local Variables

```java
public class Account {
    public String id;
    public String name;
    public double balance;

    public void display(){
        System.out.println("\tAccount information");
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Balance: " + balance);
        System.out.println();
    }
    public double balanceInDollars() {
        double balanceDollars;
        balanceDollars = balance * 3.75;
        return balanceDollars;
    }
}
```

```java
public class AccountTest {
    public static void main(String[] args) {

    Account acc1 = new Account();
    acc1.id = "1111";
    acc1.name = "Mohammad";
    acc1.balance = 3000;
    acc1.display();
    Account acc2 = new Account();
    acc2.id = "2222";
    acc2.name ="Saad";
    acc2.balance = 1000;
    acc2.display();

    double balanceDollars;
    balanceDollars = acc1.balanceInDollars();
    System.out.println("Balance of " + acc1.name
    + " in dollars is "+ balanceDollars);
    }
}
```

Two different variables

Local Variables

```
public class X {

    public int n;
    public void Y(){
        int n=10;
    }
    public void Z() {
        int n=50;
    }
}
```

```
public class Test {
    public static void main(String[] args)
    {

        X x = new X();
        x.n = 2;
        int n=100;
    }
}
```

Four different variables that have the same name

Blocks

- Recall compound statements
  - Enclosed in braces { }
- When you declare a variable within a compound statement
  - The compound statement is called a *block*
  - The scope of the variable is from its declaration to the end of the block
- Variable declared outside the block usable both outside and inside the block

Parameters of Primitive Type

```
public class Account {
    public String id;
    public String name;
    public double balance;

    public double credit(double amount){
        balance+= amount;
        return balance;
    }
    public double debit(double amount){
        if(amount <= balance)
                balance-=amount;
        else
                System.out.println("Amount exceeded balance");

        return balance;
    }
    // the rest of the previously defined methods
}
```

- Note that both credit and debit methods take one parameter which is double
- The **formal** parameter is `amount`

Parameters of Primitive Type

```
public class AccountTest {
    public static void main(String[] args) {

        Account acc1 = new Account();
        acc1.id = "1111"; acc1.name = "Mohammd";
        acc1.balance = 3000;
        double newBalance = acc1.credit(1000);
        System.out.println("The new balance "
        + "(after calling credit) is " + newBalance);
        newBalance = acc1.debit(500);
        System.out.println("The new balance "
        + "(after calling debit) is " + newBalance);
    }
}
```

- Calling the method

```
double newBalance =
acc1.credit(1000);
```

- The **actual** parameter is *the double 1000*

Sample screen output

```
The new balance (after calling credit) is 4000.0
The new balance (after calling debit) is 3500.0
```

Parameters of Primitive Type

- Parameter names are local to the method
- When method invoked
  - Each parameter initialized to value in corresponding actual parameter
  - Primitive actual parameter cannot be altered by invocation of the method

```
double despoit = 1000;
double newBalance = acc1.credit(deposit); // deposit won't change
```

- Automatic type conversion performed

```
byte -> short -> int ->
      long -> float -> double
```

Passing +1 Parameters of Primitive Type

```
public class X {

   public double n;
   public void Y(int i, int j){
      System.out.println(i + j);
   }
   public void Z(double i) {
      System.out.println(n + i);
   }
}
```

```
public class Test {
   public static void
   main(String[] args) {

      X x = new X();
      x.n = 2;
      x.Y(5,6);
      int t1= 1, t2 = 3;
      x.Y(t1,t2);
      x.Z(x.n);

   }
}
```

```
11
4
4.0
```

The use of the Keyword `this`

```java
public class X {

    public double n;
    public void Y(int i, int j){
        System.out.println(i + j);
    }
    public void Z(double n) {
        System.out.println(this.n + n);
    }
}
```

```java
public class Test {
    public static void
    main(String[] args) {

        X x = new X();
        x.n = 2;
        x.Y(5,6);
        int t1= 1, t2 = 3;
        x.Y(t1,t2);
        x.Z(6);

    }
}
```

```
11
4
8.0
```