CS 187: Programming with Data Structures (Spring 2010)

Lecture 3: Java Review II

Rui Wang
Today

• Class, reference, and instantiation
• Inheritance, accessibility
• Exception and error handling
• Input and output
• GUI

• Walk through of Assignment 1
Class

• Class
  – What is a class?
  – What’s the difference between Class and Object?
Class

• **Class**
  – Definition of a group of entities which contain both variables (data elements) and the associated operations (methods).
  – Think of it as a package that packs both variables and associated methods
  – A class is only a description, it does not yet create an object.
  – An object is created via instantiation:

    ```java
    Apple oneapple = new Apple();
    ```
Reference

- **Referencing Objects:**

  ```java
  String msg;
  msg = new String("hello world");
  String welcome = msg;
  ```

  A reference assignment does not create a new object (e.g. allocate memory space) – it merely assigns the pointer to an existing object.

  An object can have multiple references:

  ```java
  String hello = msg;
  ```
Instantiation

• **Instantiate** an Object

```java
String welcome = new String(msg);
```

Instantiation creates a new object (e.g. allocate memory space) and performs initialization.

Compare:

```java
String y = new String("abc");
String z = "def";
String w = z;
```
Constructors

- **Constructors** are methods that are called when instantiating an object. The main purpose is to initialize relevant variables.

```java
Class Apple {
    private float value;
    Apple() { value=0.0f; }
    Apple(float v) { value = v; }
    ... ...
}
```
Constructors

• **Constructors** must have the same name with the class name; they must NOT have return value; they can be overloaded.

• **The default constructor**

  ```java
  String z = new String();
  ```

  Compare the above with:

  ```java
  String z;
  ```
Nested Class Definition

• You can certainly define a class within a class

class Apple {
    public class AppleTaste {
        ...
    }
    Apple() { ... }
    Apple(float v) { ... }
}

Static Variables / Methods

• Some variables / methods are defined as static:

```java
class Apple {
    public static int value;
    public static void main(…)
    ...
}
```

How are they different from other variables / methods?
Static Variables / Methods

• Static variables exist (are allocated in memory) without any class instantiation.
  – In contrast, other variables do not exist until you have an actual object (instance).

• All class instances refer to the same static variables (i.e. they exist globally)
  – In contrast, other variables have unique local copies for each different object.

• Example:
  ```java
  Math.PI;
  ```
Static Variables / Methods

• Static methods may be called without any class instantiation
  – In contrast, other methods cannot be called until you have an actual object (instance)

• Static methods cannot call non-static methods, or refer to non-static variables.
  – Non-static variables / methods do not even exist if you don’t have an instance yet.
  – What about the reverse?
Accessibility / Visibility

• Access to variables or methods respects the declared accessibility (visibility)
  – **public** = always accessible.
  – **protected** = accessibly only in the class and any inherited class
  – **private** = accessible only in the class itself.
Accessibility / Visibility

• Analogy: think of families and secrets
  – **public** = known facts by your neighbors
  – **protected** = secrets protected by family members (not known to your neighbors)
  – **private** = secrets owned by individuals (not even shared among family members)

• Think about the prelim exam question.
Inheritance

• You can define a class by inheriting from another class:

```java
class FujiApple extends Apple {
    public String origin;
    ...
}
```

The FujiApple class automatically inherits variables / methods defined in the Apple class.
Inheritance

• The FujiApple class automatically inherits variables / methods defined in the Apple class
  – However, remember that only **public** and **protected** variables defined in the parent class are accessibly in the inherited class.
  – **private** variables are not accessible.
Parameter Passing

• You often need to provide parameters (arguments) when calling a method.

• Java passes parameters using call-by-value
  1. For a primitive type (int, double...), the value is passed to the method being called.
     – This means the method cannot modify the original argument
Parameter Passing

• Java passes parameters using call-by-value
  2. For a class type parameter, the value being passed is the reference to an object.
    – This means the method can actually modify class members
Parameter Passing

• Example 1:

```java
public static void modify(int val) {
    val = 5;
}

public static void main() {
    int a = 10;
    modify(a);
    System.out.println(a);
}
```
**Parameter Passing**

- **Example 2:**

```java
public static void modify(Point val) {
    val.x = 5;
}

... ...

public static void main() {
    Point a = new Point(0,0);
    modify(a);
    System.out.println(a.x);
}
```
Parameter Passing

• Example 3:

```java
public static void modify(Point val) {
    val = new Point(5,5);
}

... ...

public static void main() {
    Point a = new Point(0,0)
    modify(a);
    System.out.println(a.x);
}
```
The \textit{Math} Class

- \textit{Math} class defines many useful math functions.
  - abs, min, max, floor, ceil ...
  - log, pow, sin, cos, tan, sqrt ...
Exception and Error Handling

- Exceptions provide a way to handle errors (often caused by I/O, such that the program cannot continue)
- **try-catch-finally** sequence

```java
try {
    // statements that perform I/O
} catch (IOException ex) {
    System.out.println("Error occurred");
    System.exit(1);
} finally {
}
```
Exception and Error Handling

• A lot of methods, especially I/O related, require you to handle exceptions.
• You either have to use try-catch to explicitly handle the exception, or you can use the throws clause to defer the handling to the calling method.
• Eventually an exception must be handled somewhere; otherwise the compiler will complain about un-checked exceptions.
Exception and Error Handling

• Exception can be a convenient way to replace messy nested if statements:

```java
Step A
if (Step A successful) {
    Step B
    if (Step B successful) {
        Step C
        if (Step C unsuccessful) report error in Step C
    } else {
        report error in Step B
    }
} else {
    report error in Step A
}
```
Exception and Error Handling

- Exception can be a convenient way to replace messy nested if statements:

```java
try {
    Step A
    Step B
    Step C
} catch (exception indicating C filed) {
    report error in Step C
} catch (exception indicating B failed) {
    report error in Step B
} finally {
    report error in Step A
}
```
Input/Output

• InputOutput.java
  – Input your name, print out your name
  – Input your age, print out our age
  – Input your height, print out your height
  – Using Scanner.java
import java.util.*;

class InputOutput {
    public static void main(String args[]) {
        Scanner S = new Scanner(System.in);
        String name;
        System.out.print("enter your name: ");
        name = S.next();
        System.out.println("Your name is: "+ name);
    }
}
int age;
System.out.print("enter your age: ");
age = S.nextInt();
System.out.println("Your age is: " + age);

float height;
System.out.print("enter your height(feet): ");
height = S.nextFloat();
System.out.println("Your height is: " + height + " feet");
}
Simple GUI

- **JOptionPane**

  ```java
  String name = JOptionPane.showInputDialog(“Enter your name”);
  ```
  - Will pop up a dialog window with an input field
  - If you click on OK, the input will be stored in `name`.
  - If you click on Cancel, `name` will be null.
Simple GUI

• **JOptionPane**
  
  `JOptionPane.showMessageDialog (null, "Your name is " + name);
  
  – Will pop up a message window displaying the message.
More Complex GUI

• JOptionPane

```java
String[] choices={"bacon","sausage","egg","apple"};
int selection = JOptionPane.showOptionDialog
    (null, "Make a choice",
     "Menu",
     JOptionPane.YES_NO_CANCEL_OPTION,
     JOptionPane.QUESTION_MESSAGE,
     null, choices, choices[0]);
```

– Will pop up a dialog window with buttons indicating difference choices you can make.
Coming up

• Assignment 1 is due by Thursday this week

• Discussion section tomorrow
  – Review of the preliminary test
  – More details on assignment 1
Logistics

- Overriding issues
- EDLAB (LGRT 225)
- Textbook
Logistics

• General advice regarding questions:
  – Please try to avoid emailing your questions directly to the TA or me
  – Especially avoid sending code directly to us
  – Post your questions on discussion group
  – Make full use of the TA office hours
Logistics

• In general, for each assignment:
  – Starter code will be provided
  – You are supposed to modify the starter code to implement required functionality
  – Places that need your work are usually marked very clearly (i.e. // fill your code here)
  – Submit the modified .java files and the compiled .class files
Assignment 1

• Starter code was updated on Thursday right after class, make sure you have the current version.
• FAQ added
• Detailed grading criteria added
Assignment 1

• A complete walk-through
  – Assume I start everything from scratch...

• Explanation, example

• Demo

• Grading
  – Produce correct results for any positive window size and any number of positive input values
  – We are not picky about error handling