Teaching Week 2 'Recap'

covering

** Inheritance

** Getters/Setters

** Abstract Classes

Chapters 4-5 (section 4.2, 5.1+5.8) – "Core Java" book
Chapter 4, 7, 8 – "Head First Java" book
Chapters 8, 11, 15 (sections 8.9, 11.1-11.5, 15.2) – "Introduction to
Java Programming" book
Chapter 3 – "Java in a Nutshell" book

Queen Mary
University of London

EBU4201 © 2023/24

Example (1/6): Rectangle Class

```java
class Rectangle { // instance variables (attributes)
  int width;
  int height;

  /* draw the rectangle on the terminal */
  void draw() {
    for (int i = 0; i < height; i++) {
      for (int j = 0; j < width; j++) {
        System.out.print("*");
      } // end for
      System.out.println("");
    } // end for
  } // end draw() method

  /* returns the area of the rectangle */
  int area() {
    return width * height;
  } // end area() method
} // end class Rectangle
```

Example (2/6): Rectangle Class Description

- Declares a class **Rectangle** with two attributes, **width** and **height**, both of which are integers.

- These instance variables are not *qualified,* so are assumed to be *public.*

  (This is not the whole truth – they actually fall into that default access we talked about!)

- There are two operations that can be performed on an instance of the **Rectangle** class:

  - **draw()**, which tells the **Rectangle** to draw itself;

  - **area()**, which causes the **Rectangle** to compute and return its own area.

Queen Mary
University of London

Example (3/6): Using the Rectangle Class

```java
class RectangleTest {
  public static void main (String[] argv) {
    Rectangle r1;
    r1 = new Rectangle();
    r1.draw();
    System.out.println("area is " + r1.area());
  } // end main() method
} // end class RectangleTest
```
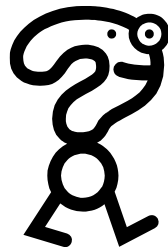
1. What will this program do?
2. How can it be improved?

Example (4/6): Improving the Test Program

```java
class RectangleTest {
  public static void main (String[] argv) {
    // 1: declare (create) a rectangle variable
    Rectangle r1;
    // 2: create *instance of* a rectangle
    r1 = new Rectangle();
    // 3: set the width of r1
    r1.width = 10;
    // 4: set the height of r1
    r1.height = 20;
    // 5: tell r1 to draw itself
    r1. draw();
    // 6: print the area of r1
    System.out.println("area is " + r1.area());
  } // end main() method
} // end class RectangleTest
```

Example (5/6): Analysis of Test Program

- Line (1): Declares a variable **r1** to be a **Rectangle**.
  - At this point, no **Rectangle** object has been created.
  - The default value for an object variable is a special value called **null**, which means "no object".

- Line (2): *Creates* a rectangle object.
  - The **new** keyword says "create me a…".
  - Call to **Rectangle()** is to the object's default *constructor*.
  - Upon creation, all instance variables in an object are initialised to their default values.

What values are **width** and **height** initialised to?

- Line (3): *Accesses* the instance variable **width**, and sets it to *10*.
  - This direct access is only possible because **width** is publicly accessible.

- Line (4): As Line (3), but *sets* the **height** instance variable.

- Line (5): Tells the **r1** object to invoke the **draw()** method.
  - The idea is very similar to the function call in C.

- Line (6): This line *invokes* the **area()** method on the **r1** object, which returns the area of **r1** (200).
  - The area is returned from the method via the **return** operation.

Access Control – 1/4

- Remember, we don't *want* to make all our instance variables and methods public – this defeats the purpose of *information hiding*.

```java
class Rectangle {
    // instance variables (attributes)
    /* the width of the rectangle in cm */
    private int width;
    /* the height of the rectangle in cm */
    private int height;

    // other code
}
```

- However, now in our **main()** program, the following will cause an error:

```java
Rectangle r1 = new Rectangle();
r1.height = 10; // ERROR HERE!
```

Access Control – 2/4

- The **area()** and **draw()** methods will still work; because they are *part* of the object, they can see its private parts …

- To explicitly make an instance variable or method public, we qualify the declaration with the **public** keyword:

```
/* returns the area of rectangle */
public int area() {
    return width * height;
} // end area() method
```

- By default, you should make all instance variables **private**.

Access Control – 3/4

- *Accessor methods: "get" for* `width`

```
/* get method for width */
public int getWidth() {
    return this.width;
} // end width() method
```

  – Simply returns the value of the variable.

  – Convention: give the *get* method same name as the variable.

- *Mutator methods: "set" for* `width`

```
/* set method for width */
public void setWidth(int i) {
    if (i >= 0) { this.width = i; }
    else { System.out.println("bad width"); } // end if-else
} // end setWidth() method
```

- Improved RectangleTest program:

```
class RectangleTest {
  public static void main(String[ ] argv) {
    Rectangle r1 = new Rectangle();
    r1.setWidth(10);
    r1.setHeight(20);
    r1.draw();
    System.out.println("area is " + r1.area());
  } // end main() method
} // end class RectangleTest
```

Queen Mary
University of London

Writing your own Constructors

- Previously to create a new rectangle and initialise it, the following code was required:

```
Rectangle r1 = new Rectangle();
r1.setWidth(10);
r1.setHeight(20);
```

- Can simplify this by writing our own constructor method:

```
public Rectangle(int w, int h) {
    setWidth(w);
    setHeight(h);
}
```

- Thus the original 3 lines of code above can be reduced to:

```
Rectangle r1 = new Rectangle(10,20);
```

Using `this`

- Sometimes, an object needs to be able to refer to itself.

  - It does this using the keyword **this**.

  - **Example**:

```java
public boolean isSquare() {
  if (this.width == this.height) {
    return true;
  }
  else {
    return false;
  }
}
```

Example (1/2): Inheritance

- We will create a **Square** class as a subclass of **Rectangle**, which *reuses* much of its code.

```java
class Square extends Rectangle {
  public Square(int w, int h) {
    if (w != h) {
      System.out.println("bad square!");
    }
    else {
      setWidth(w);
      setHeight(h);
    } // end else
  }
  public Square(int w) {
    setWidth(w);
    setHeight(w);
  }
} // end class Square
```

Queen Mary
University of London

Example (2/2): Overriding Constructors

- This class **Square** *inherits* all the instance variables and methods of the **Rectangle** class.

- **Example**:

```
Square s1 = new Square(10,10);
System.out.println("area s1 = " + s1.area());
```

- When methods or constructors in the *subclass* are provided that take the same name and parameters as those in the *superclass*, then the ones in the *subclass* are used.

- The constructor

```
public Square(int w, int h) {
   // some code

}
```

overrides the similar constructor in **Rectangle**.

Queen Mary
University of London