Lecture 4 – Reference Types

08-671
Java Programming for App Developers

September 10, 2015

Jeffrey L. Eppinger & Terry Lee

08-671 Lecture Topics

(subject to change – but only a little bit)

#1 Intro
#2 Primitive Types
#3 Java Classes
#4 Reference Types
#5 Loops & Arrays
#6 Lists & Sorting
#7 Maps
#8 File & Network I/O
#9 Swing Interfaces
#10 Swing Actions
#11 Threads
#12 Exceptions
#13 Functional Programming
#14 In-class Written Exam

* Final Exam – this will be a 3-hour programming problem

TA Office Hours

- Easier to find on Blackboard

Outline

⇢Questions

N-JAPL

Homework #3

Real-time Development Exercises ☺

Questions

Outline

✓ Questions

⇢ N-JAPL

Homework #3

Real-time Development Exercises ☺

Questions

N–JAPL

- NOT Just Another Programming Language
  - Objects
  - Runtime environments

“Object” In Olden Days (e.g., C)

Define a type

```
typedef struct rectangle {
    int width;
    int height;
} Rectangle;
```

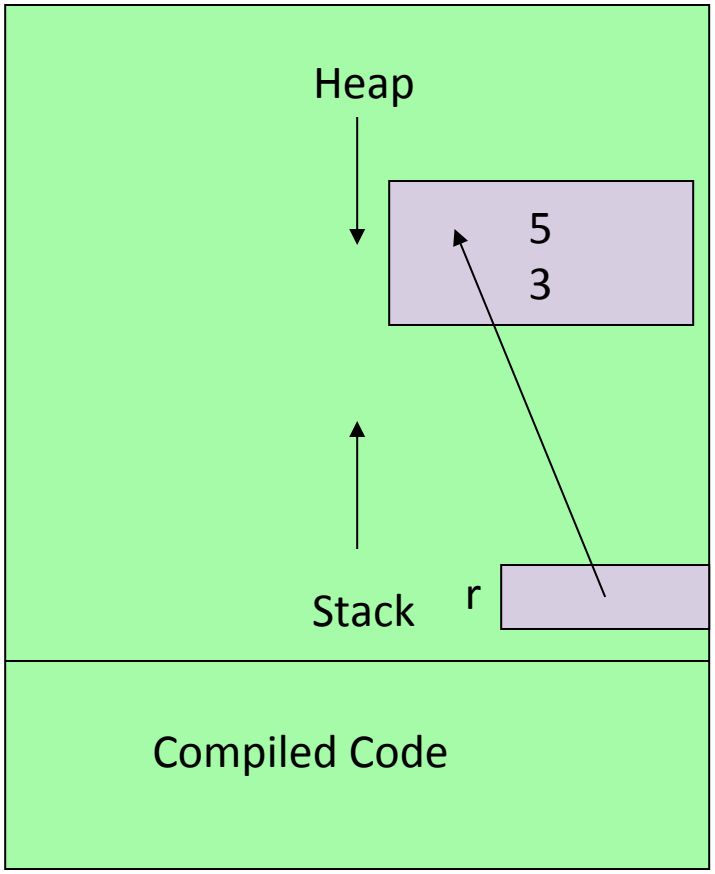
Creating the Object in Olden Days

```
Rectangle* r = (Rectangle*) malloc(sizeof(Rectangle));
r->width = 5;
r->height= 3;
```

Or we would write a method containing this code…

```
Rectangle r = createRectangle(5,3);
```

How Did It Work?



Nowadays (in Java)

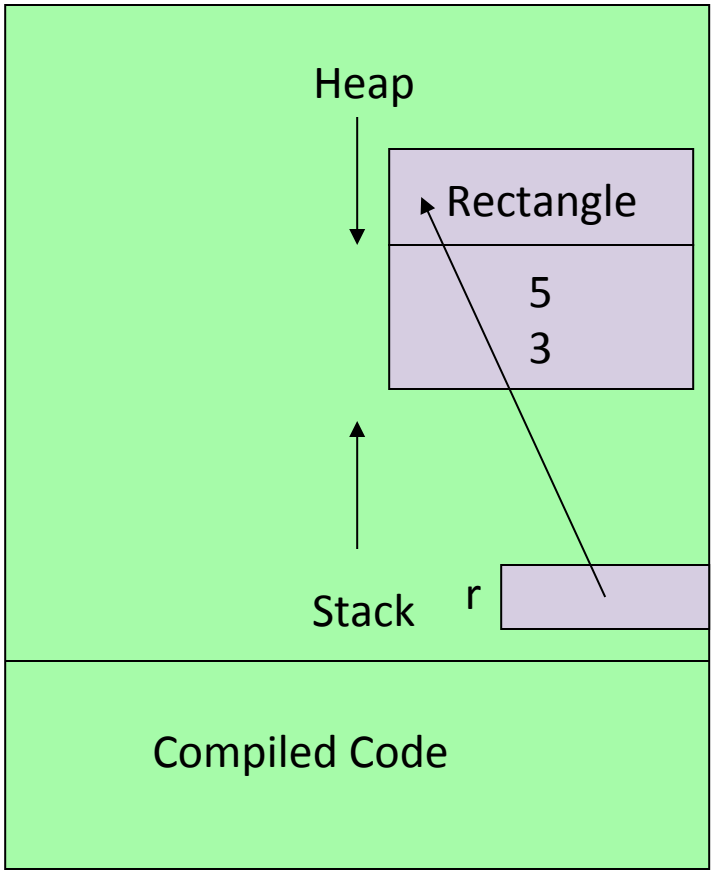
Define a class

```
public class Rectangle {
    int  width;
    int  height;
}
```

Use the "new" operator to instantiate the class

```
Rectangle r = new Rectangle();
r.width  = 5;
r.height = 3;
```

How Does It Work?



Optionally, Write a Constructor

```
public class Rectangle {
    int width;
    int height;

    // This is a constructor method…
    public Rectangle(int newWidth, int newHeight) {
        width  = newWidth;
        height = newHeight;
    }
}
```

You Must Use a Constructor

If constructors are provided, you must use one

```
Rectangle r = new Rectangle(5,3);
```

If no constructors are provided, Java generates a "default" constructor – with no arguments

```
Rectangle r = new Rectangle();
```

Advantages of Constructors

- You don't have to explain to users details of how to initialize the object – just call a constructor
- You control the initialization of the object
  - If you provide a constructor (or constructors) users must call it (or one of them)

Example

```
public class StockHolding {
    String ticker;
    int    shares;
    String name;
    float  price;


    // This is a constructor method…
    public StockHolding(String newTicker, int newShares) {
        ticker = newTicker;
        shares = newShares;
        StockQuote sq = new StockQuote(ticker);
        name   = sq.getName();
        price  = sq.getPrice();
    }
}
```

Example Test Program

```
public class StockHoldingTest {
    public static void main(String[] args) {
        StockHolding h = new StockHolding(args[0],
                                Integer.parseInt(args[1]);
        System.out.println(h.ticker + ", " +
            h.shares + ", $" + h.price + " (" +
            h.name + ")");
    }
}
```

Extended AddressBookEntry

```
public class AddressBookEntry {
    String firstName;
    String lastName;
    String telephoneNumber;
    String eMailAddress;
    AddressBookEntry mother;
    AddressBookEntry father;

    …
}
```

You can reference instances of your object

Access Modifiers

- There are "four" access modifiers
- For now, lets look at the usual two

  `public` – Everyone can access it

  `private` – Only this class (file) can access it
- We like to declare instance variables to be private to prevent changes by "others" (users of our class)
- We also declare some methods (and variables) to be private so as not to expose the internal implementation details of our class
  - It's easier to use if you don't need to know
  - It's easier to change if you didn't depend on the specific implementation

Example: java.lang.String

- We've used Strings
- What are they?
- Let's look at the Java Doc for String

  http://docs.oracle.com/javase/8/docs/api

Shape

```
public class Shape {
    private double area;

    public Shape(double newArea) {
        area = newArea;
    }

    public double getArea() { return area; }

    public String toString() {
        return "Shape(area=" + area + ")";
    }
}
```

Subclasses & Inheritance

- Subclass using "**`extends`**" when declaring a class
- Example: Many shapes

  `Rectangle.java`
  `Circle.java`
  `Square.java`

- Why subclass?
  - You can be just like your ancestors, but a few things different
    - Extra functions?
    - Overridden methods (not to be confused with overloaded methods)
    - Hidden variables
    - Different Constructors

Rectangle

(rewritten as subclass of Shape, using private instance variables)

```
public class Rectangle extends Shape {
    private double width;
    private double height;

    public Rectangle(double newWidth, double newHeight) {
        super(newWidth * newHeight);
        width  = newWidth;
        height = newHeight;
    }

    public double getHeight() { return height; }
    public double getWidth()  { return width;  }

    public String toString() {
        return "Rectangle(width=" + width + ", height=" + height);
    }
}
```

Must Call the Super Class Constructor

- The first line of a subclass's constructor must be a call to the superclass constructor
  - If not there, you'd get an error if it has args
- If the super class has a no-args constructor Java will automatically generate the call
  - Examples:
    - MyEntry is a subclass of AddrBookEntry
    - Shape (and everything else) is a subclass of Object

Circle

```
public class Circle extends Shape {
    private double radius;

    public Circle(double newRadius) {
        super(Math.PI * newRadius * newRadius);
        radius = newRadius;
    }

    public double getRadius() { return radius; }

    public String toString() {
        return "Circle(radius=" + radius + ")";
    }
}
```

Square

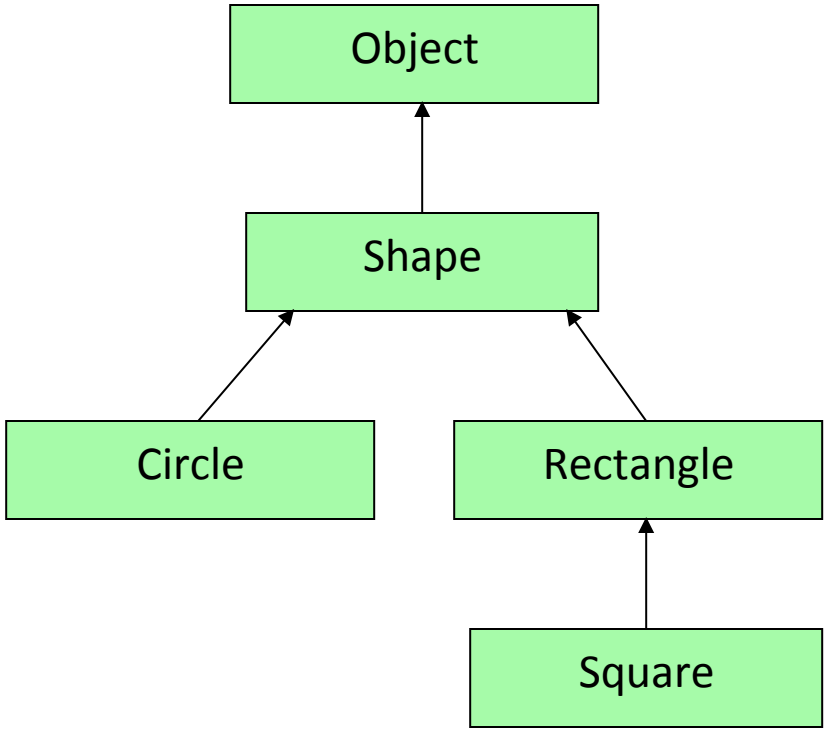
```
public class Square extends Rectangle {
    private double side;

    public Square(double newSide) {
        super(newSide,newSide);
        side = newSide;
    }

    public double getSide() { return side; }

    public String toString() {
        return "Square(side=" + side + ")";
    }
}
```

The Class Hierarchy



The `toString()` method

- It's defined in `java.lang.Object`
  - You can call it on any object
- If a class doesn't like what is printed, it can implement it's own `toString()` method
  - This "overrides" the superclass `toString()`

2nd Example Test Program

```
public class ShapeTest {
    public static void main(String[] args) {
        Circle c = new Circle(3);
        System.out.println(c + " area=" + c.getArea());

        Rectangle r = new Rectangle(5,3);
        System.out.println(r + " area=" + r.getArea());

        Shape s = new Square(3);
        System.out.println(s + " area=" + s.getArea());

        Shape s1 = c;
        Shape s2 = r;
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

3rd Example Test Program

```
public class ToStringTest {
    public static void main(String[] args) {
        Object o1 = new Square(3);
        Object o2 = new StockHolding("GOOG",100);

        System.out.println(o1);
        System.out.println(o2);
    }
}
```

Casting

Convert from one type of reference to another

No explicit cast when going up the class hierarchy

```
Square s = new Square();
Object o = s;
```

Explicit cast is required going down the class hierarchy

```
Shape shape = shapeMaker(…);
Rectangle r = (Rectangle) shape;
if (r instanceof Square) {
    Square s = (Square) r;
}
```

Additional Functionality for Classes

- Constructors
  - Create new objects using **`new`** keyword
  - Allocates space for the (instance) variables
- Method Declaration
  - Overriding (same name and params as in superclass – e.g., toString())
  - Overloading (same name, same class, different params – e.g., println())
- Member Modifiers
  - Access (**`public`**, **`private`**, etc)
  - Others later on (**`static`**, **`final`**, **`abstract`**)
- Subclasses (**`extends`**)
- Interfaces, later on (**`implements`**)
- Packages, later on

Giving Back Storage

- When you're finishing with an object, how do you give it back?
  - This is important for long-running programs

Reclaiming Storage in “Olden” Days

- You must call the “free” function to return allocated space to the pool:

```
Rectangle r = (Rectangle*) malloc(sizeof(Rectangle));
...
free(r);
```

- Who’s responsible for doing this?

Nowadays (in Java)

Java implements "Garbage Collection"

- Java figures out, itself, when you are finished with the storage!
- There is no "free" function

How does this work?

- Java keeps "track" of all references to objects
- Objects with "no references" are "freed"

Outline

✓ Questions

✓ N-JAPL

⇢ Homework #3

Real-time Development Exercises ☺

Questions

Homework 3

- Homework #3 will posted after class
- Due Tues, 9/15 at 23:59
- Next week we will be covering:
  - Conditionals
  - Loops & Arrays
  - Much more about methods and classes
- You can do Homework #3 without conditionals, loops, arrays, or defining any methods of your own
  - But it's okay if you use these constructs in your solutions
  - I'm expecting you to do it using only mathematical expressions and string manipulation

Comments

- Style checker will require comments for each class and each method
  - Class comment must have an `@author` tags
  - Method comment must have `@param` and `@return` tags
- We will provide these comments in our examples when we post them tonight

Outline

- ✓ Questions
- ✓ N-JAPL
- ✓ Homework #3
- ⇢ Real-time Development Exercises ☺
- Questions

Real-time Development Exercises ☺

- Let's write a sample program
- Let's do this with Eclipse
  - Download from www.eclipse.org
  - I'm using the Eclipse Standard
  - It has Windows, Linux, and Mac support
  - If you're using another version of Eclipse – it's OK
    - I'll be using Eclipse J2EE in my Mini 2 class

Outline

- ✓ Questions
- ✓ N-JAPL
- ✓ Homework #3
- ✓ Real-time Development Exercises ☺
- ⇢ Questions

Sample Final Exam Question

- What's the difference between a primitive type and a reference type?
  - Primitive types used fixed storage
    - Variables of primitive type contain their values
  - Reference types are objects (and arrays) that use varied amounts of storage
    - Amounts that are different for each object (and array)
    - Variables of reference type "reference" (point to) their values (which are instances of their type)
  - (We'll continue to enhance this definition when we cover methods and arrays next week)

Sample Final Exam Question

- What is the advantage of using constructors?

Recitation Tomorrow

At 1:30pm in DH A302

- More with Eclipse
- More about objects
- More on JavaDoc
- Q&A
  - Bonus Prizes for Questions?
  - No prizes for HW3
- Quiz