Slides with a blue heading are for "homework".

Tutorial: Teaching Week 3 SELECTED SOLUTIONS

Topics:

- Abstract Classes & Polymorphism
- References to Objects & Understanding the Stack
- Garbage collection
- Interfaces
- Overloaded Constructors & constructor chaining
- **`static`** & **`final`**: methods and variables
- **`null`** references / Strings
- GUI

Sources of some questions:

- ✓ "Introductory Java Programming" book
- ✓ "Head First Java" book
- ✓ Java tutorial from http://docs.oracle.com

Queen Mary University of London

Abstract Classes

```
abstract class Card {
  String recipient;
  public abstract void greeting();
}
```

Card.java

```
public class BirthdayCard extends Card {
  int age;
  public BirthdayCard(String r, int years) {
    recipient = r;
    age = years;
  }
  public void greeting() {
    System.out.println("Dear " + recipient + ",\n");
    System.out.println("Happy " + age + "th Birthday!\n\n");
  }
}
```

BirthdayCard.java

```
public class HolidayCard extends Card {
  public HolidayCard(String r) { recipient = r; }
  public void greeting() {
    System.out.println("Dear " + recipient + ",\n");
    System.out.println("Season's Greetings!\n\n");
  }
}
```

HolidayCard.java

```
public class ValentineCard extends Card {
  int kisses;
  public ValentineCard(String r, int k) {
    recipient = r;
    kisses = k;
  }
  public void greeting() {
    System.out.println("Dear " + recipient + ",\n");
    System.out.println("Love and Kisses,\n");
    for (int j=0; j < kisses; j++) System.out.print("X");
    System.out.println("\n\n");
  }
}
```

ValentineCard.java

```
public class CardTester1 {
  public static void main(String[] args) {
    String me = args[0];

    HolidayCard hol = new HolidayCard(me);
    hol.greeting();

    BirthdayCard bd = new BirthdayCard(me, 18);
    bd.greeting();

    ValentineCard val = new ValentineCard(me, 3);
    val.greeting();
  }
}
```

CardTester1.java

What is the output?

```
> java CardTester1 George
Dear George,

Season's Greetings!


Dear George,

Happy 18th Birthday!


Dear George,

Love and Kisses,

XXX
```

Queen Mary University of London

Polymorphism

```
public class CardTester2 {
  public static void main(String[] args) {
    // Invokes a HolidayCard greeting().
    Card card = new HolidayCard("Amy");
    card.greeting();

    // Invokes a ValentineCard greeting().
    card = new ValentineCard("Bob", 3);
    card.greeting();

    // Invokes a BirthdayCard greeting().
    card = new BirthdayCard("Cindy", 17);
    card.greeting();

  }
}
```

CardTester2.java

```
> java CardTester2
Dear Amy,

Season's Greetings!


Dear Bob,

Love and Kisses,

XXX


Dear Cindy,

Happy 17th Birthday!
```

What is the output?

References to Objects

```
public class CardTester3 {
  public static void main(String[] args) {
    Card c;
    ValentineCard v;
    BirthdayCard b;
    HolidayCard h;

    c = new ValentineCard("Debby", 8);    // OK
    b = new ValentineCard("Elroy", 3);    // Wrong
    v = new ValentineCard("Fiona", 3);    // OK
    h = new BirthdayCard("Greg", 35);     // Wrong
  }
}
```

CardTester3.java

Which statements are OK?

Interfaces (1/3)

Consider an interface named **Colorable**, as follows:

```
public interface Colorable { public void howToColor(); }
```

- Create a class named **Square** that extends **GeometricObject** *and* implements **Colorable**.
- Implement **howToColor()** to display a message on how to colour the square.
- Create an additional class to test the creation of a **Square** and its method **howToColor()**.

```
public abstract class GeometricObject {
  // some methods and instance variables

  public abstract double findArea();
  public abstract double findPerimeter();
}
```

```
public class Square extends GeometricObject
                    implements Colorable {
  private double side;
  public Square(double side) { this.side = side; }
  public void howToColor() { System.out.println("Colour all four sides."); }
  public double findArea() { return (this.side * this.side); }
  public double findPerimeter() { return (4 * this.side); }
}

public class Test {
  public static void main(String[] args) {
    Square square = new Square(2);
    square.howToColor();
  }
}
```

Interfaces (2/3)

- Answer the following questions:
  - Identify what is wrong with the **`interface`** below.

```
public interface SomethingIsWrong {
  void aMethod(int aValue) {
    System.out.println("Hi Mom");
  }
}
```

  - Can an interface be given the **`private`** access modifier?

Answers:

- Interfaces **can not** have *method implementations*; they *can only have method declarations and constants*.
- **No** – if that was the case, then the interface could never be used.

Interfaces (3/3)

Identify the statements below about interfaces, that are TRUE.

a. Interfaces do not allow for multiple inheritance at design level.
b. Interfaces can be extended by any number of other interfaces.
c. Interfaces can extend any number of other interfaces.
d. Members of an interface are never `static`.
e. Methods of an interface can always be declared `static`.

- True – (b) and (c)
- (a) Interfaces do not have any implementations and only permit multiple interface inheritance.
- (b+c) An interface can extend any number of other interfaces and can be extended by any number of interfaces.
- (d+e) Fields in interfaces are always **static** and method prototypes in interfaces are never **static**.

Identify the field declarations that are legal in the body of an interface.

a. `public static int answer = 10;`
b. `int answer;`
c. `final static int answer = 10;`
d. `public int answer = 10;`
e. `private final static int answer = 10;`

- Legal – (a), (c), and (d)
- Fields in interfaces declare named constants, and are always **public**, **static**, and **final**.
- None of these modifiers are mandatory in a constant declaration.
- All named constants must be explicitly initialised in the declaration.

Constructors

- Identify the constructors in class **SonOfBoo** that are not legal.

```
public class Boo {
  public Boo(int i) { }
  public Boo(String s) { }
  public Boo(String s, int i) { }
}

class SonOfBoo extends Boo {
  public SonOfBoo() { super("boo"); }
  public SonOfBoo(int i) { super("Fred"); }
  public SonOfBoo(String s) { super(42); }
  public SonOfBoo(String a, String b, String c) { super(a, b); }
  public SonOfBoo(int i, int j) { super("man", j); }
  public SonOfBoo(int i, int x, int y) { super(i, "star"); }
}
```

(1)

(2)

(3)

OK – calls (2)

OK – calls (2)

OK – calls (1)

OK – calls (3)

Not OK

Not OK

Garbage Collection

- Identify the lines of code that, if added to the program at point **X** would cause exactly one more object to be eligible for the **Garbage Collector**.

```
copyGC = null;
gc2 = null;
newGC = gc3;
gc1 = null;
newGC = null;
gc4 = null;
gc3 = gc2;
gc1 = gc4;
gc3 = null;
```

Answer

```
public class GC {
  public static GC doStuff() {
    GC newGC = new GC();
    doStuff2(newGC);
    return newGC;
  }
  public static void main(String[] args) {
     GC gc1;
     GC gc2 = new GC();
     GC gc3 = new GC();
     GC gc4 = gc3;
     gc1 = doStuff();
     X
     // call more methods
  }
  public static void doStuff2(GC copyGC) {
    GC localGC;
  }
}
```

Note that variable **gc1** is not initialised to a default value here, because it is a local variable.

Constructor Chaining

- Determine the order in which the constructors execute in this example.

Answer:
C1, then C2, then C3

```java
//Should be in C1.java
public class C1 {
    public C1() {
        System.out.println("1");
    }
}
//Should be in C2.java
public class C2 extends C1{
    public C2() {
        super();
        System.out.println("2");
    }
}
//Should be in C3.java
public class C3 extends C2 {
    public C3() {
        System.out.println("3");
    }

    public static void main(String args[]) {
        //Q: What list of numbers will be printed?
        //   (What order are the constructors executed?)
        C3 obj = new C3();
    }
}
```

Queen Mary University of London

static and instance methods

What is wrong with the code on the right?

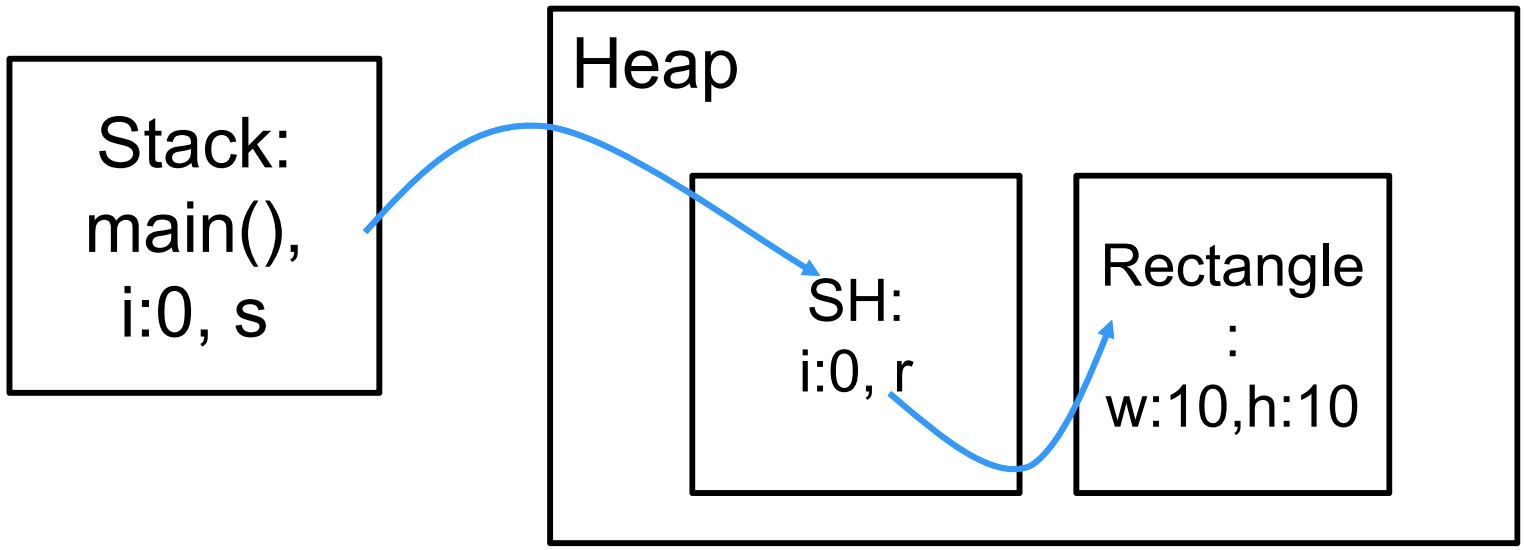
```
public class  ExampleGoneWrong
{
    public int i = 0;
    public static int s = 0;
    public void doSomethingInstance() {
        System.out.println("Instance Method");
        System.out.println(i);
        System.out.println(s);
    }
    public static void doSomethingStatic() {
        System.out.println("Static Method");
        System.out.println(i);
        System.out.println(s);
    }
    public static void main(String args[]){
        doSomethingStatic();
        doSomethingInstance();
    }
}
```

```
public class  ExampleGoneWrong
{
    public int i = 0;
    public static int s = 0;
    public void doSomethingInstance() {
        System.out.println("Instance Method");
        System.out.println(i);  //Ok. Instance variable from instance context.
        System.out.println(s);  //Ok. Static variable from instance context.
    }
    public static void doSomethingStatic() {
        System.out.println("Static Method");
//      System.out.println(i);  //Error. Instance variable from static context.
        System.out.println(s);  //Ok. Static variable from instance context.
    }
    public static void main(String args[]){
        doSomethingStatic();    //Ok. static method from static context.
//      doSomethingInstance();  //Error. Instance method from static context.

        //To use instance method/variable in static context, create an object first.
        ISA obj = new ISA();
        obj.doSomethingInstance();  //Call the instance method on the object.
        System.out.println(obj.i);  //Access the instance variable on the object.
    }
}
```

Stack and Heap Storage

- Which is on *stack* and which is on *heap*?

```
import java.awt.Rectangle;
public class SH {
    public int i = 0;
    public Rectangle r = new Rectangle(10,10);
}

public class SHTest {
    public static void main(String args[])
    {
        int i = 0;
        SH s = new SH();
    }
}
```



`final` variables and *pass-by-value*

What is the output?

```
>java FinalPass
[I@272d7a10
A: 0, 1
[I@272d7a10
B: 0, 1
[I@272d7a10
C: 1, 1
[I@272d7a10
D: 1, 0
[I@272d7a10
E: 1, 0
[I@272d7a10
F: 9, 9
```

```
public class FinalPass {
    final static int[] array1 = new int[] {0, 1};
    final static int[] array2 = new int[] {1, 0};

    public static void main(String[] args) {
        System.out.println(array1);
        System.out.println("A: " + array1[0] + ", " + array1[1]);

        array1=array2;
        System.out.println(array1);
        System.out.println("B: " + array1[0] + ", " + array1[1]);

        array1[0]++;
        System.out.println(array1);
        System.out.println("C: " + array1[0] + ", " + array1[1]);

        array1[0] = array2[0];
        array1[1] = array2[1];
        System.out.println(array1);
        System.out.println("D: " + array1[0] + ", " + array1[1]);

        method1(array1);
        System.out.println(array1);
        System.out.println("F: " + array1[0] + ", " + array1[1]);
    }

    public static void method1(int[] array){
        System.out.println(array);
        System.out.println("E: " + array1[0] + ", " + array1[1]);
        array[0] = 9;
        array[1] = 9;
    }
}
```

Strings (1/3)

- Consider the following string:

```
String hannah = "Did Hannah see bees? Hannah did.";
```

  - What is the value displayed by the expression `hannah.length()`?
  - What is the value returned by the method call `hannah.charAt(12)`?
  - Write an expression that refers to the letter `b` in the `String` referred to by `hannah`.

Answers:
32; **e**; **hannah.charAt(15)**

Write a program that *computes your initials* from your full name and *displays them*.

```
public class ComputeInitials {
  public static void main(String[] args) {
    String myName = “Arthur C. Clarke”;
    StringBuffer myInitials = new StringBuffer();
    int length = myName.length();
    for (int i = 0; i < length; i++) {
      if (Character.isUpperCase(myName.charAt(i))) {
        myInitials.append(myName.charAt(i));
      }
    }
    System.out.println("My initials are: " + myInitials);
  }
}
```

Output → My initials are: ACC

Strings (2/3)

- In the program below, what is the value of **`result`** after each *numbered line* executes?

```
  public class ComputeResult {
    public static void main(String[] args) {
      String original = "software";
      StringBuilder result = new StringBuilder("hi");
      int index = original.indexOf('a');
/*1*/ result.setCharAt(0, original.charAt(0));
/*2*/ result.setCharAt(1, original.charAt(original.length()-1));
/*3*/ result.insert(1, original.charAt(4));
/*4*/ result.append(original.substring(1,4));
/*5*/ result.insert(3, (original.substring(index, index+2) + " "));
      System.out.println(result);
    }
  }
```

Answers:

```
/*1*/ si
/*2*/ se
/*3*/ swe
/*4*/ sweoft
/*5*/ swear oft
```

Strings (3/3)

Which two statements are TRUE?

a. `String` objects are immutable.
b. Subclasses of the `String` class can be mutable.
c. All wrapper classes are declared `final`.
d. All objects have a `private` method named `toString()`.

What is the output of the program below?

```
public class ExampleStrings {
  public static void main(String[] args) {
    String str1 = "ab" + "12";
    String str2 = "ab" + 12;
    String str3 = new String("ab12");
    System.out.println((str1==str2) + " " + (str1==str3));
  }
}
```

True: (a) and (c)

c) The **String** class and all wrapper classes are declared **final** and, therefore, cannot be extended.

d) The **toString()** method is declared **public** in the **Object** class.

b) **String** objects are immutable and therefore, cannot be modified.

The program will print **true false** when run.

The constant expressions "ab" + "12" and "ab" + 12 will, at compile time, be evaluated to the string-valued constant "ab12".

Both variables **str1** and **str2** are assigned a reference to the same interned **String** object containing "ab12". The variable **str3** is assigned a new **String** object, created using the **new** operator.

GUI (1/4)

- The following program is supposed to display a button in a frame, but nothing is displayed. What is the problem?

```
import javax.swing.*;

public class Test extends JFrame {
  public Test() {
    getContentPane().add(new JButton("OK"));
  }

  public static void main(String[] args) {
    JFrame frame = new JFrame();
    frame.setSize(100,200);
    frame.setVisible(true);
  }
}
```

`Test frame = new Test();`

GUI (3/4)

- Choose the layout manager(s) most naturally suited for the following layout description, an example of which is given below: "the container has a row of components that should all be displayed at the same size, filling the container's entire area".

a. `FlowLayout`
b. `GridLayout`
c. `BorderLayout`
d. Options *a* and *b*.

Note: You can assume that the container controlled by the layout manager is a `JPanel`.

Answer: *b*. This type of same-size layout (whether in a row, a column, or a grid) is what **GridLayout** is best at.

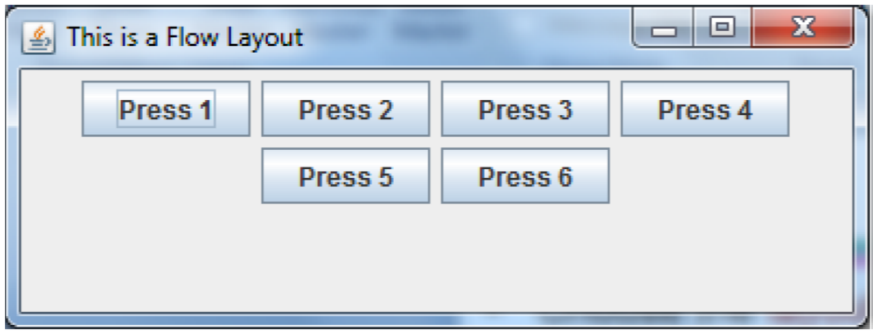
Extra Question: What would be the implementation of this layout (in *b*)?
Answer:

```
JFrame frame = new JFrame("Layout2");
JPanel myPanel = new JPanel(new GridLayout(1,0));=
myPanel.add(createComponent("Component 1"));
myPanel.add(createComponent("Component 2"));
myPanel.add(createComponent("Component 3"));
myPanel.add(createComponent("Component 4"));
frame.setContentPane(myPanel);
```

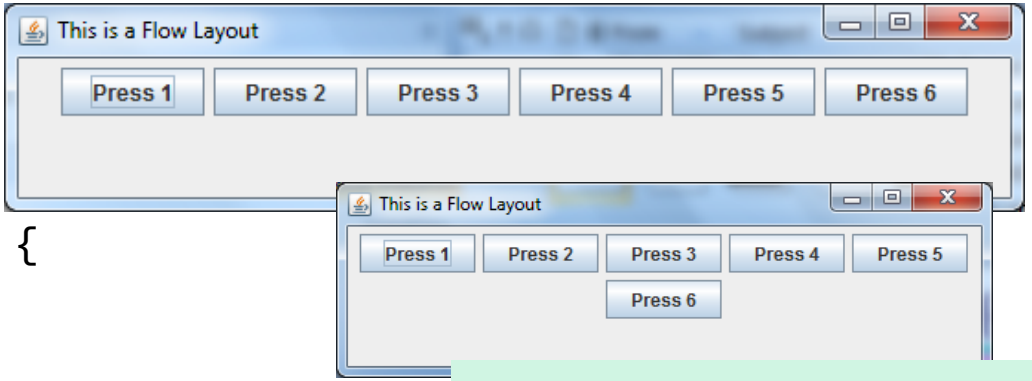
GUI (4/4)

The GUI below uses a **`FlowLayout`** manager to arrange the display of the 6 buttons.

- Write the Java code that generates this GUI.
- What would happen to the displayed GUI if it was resized into a bigger window?




```
import javax.swing.*; import java.awt.*;
public class TryFlowLayout {
  public static void main(String[] args) {
    int windowWidth = 400;
    int windowHeight = 150;
    JFrame aWindow = new JFrame("This is a Flow Layout");
    aWindow.setBounds(100, 100, windowWidth, windowHeight);
    aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    FlowLayout flow = new FlowLayout();
    Container content = aWindow.getContentPane();
    content.setLayout(flow);
    for (int i = 1; i <= 6; i++) content.add(new JButton("Press " + i));
    aWindow.setVisible(true);
  }
}
```



Possible outcomes …