COMP 401: CONSTRUCTORS AND POINTERS

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
public class ALoopingFactorialSpreadsheet {
    int number;
    long factorial;
    public int getNumber() {
        return number;
    }
    public void setNumber(int newVal) {
        number = newVal;
        factorial = Factorials.loopingFactorial(number);
    }
    public long getFactorial() {
        return factorial;
    }
}

ALoopingFactorialSpreadsheet factorial1 = new ALoopingFactorialSpreadsheet();
ALoopingFactorialSpreadsheet factorial = new ALoopingFactorialSpreadsheet();
ALoopingFactorialSpreadsheet factorial = new ALoopingFactorialSpreadsheet();
factorial1.setNumber(2);
factorial2.setNumber(2);
Another Class with Properties

```java
public class ABMISpreadsheet {
    double height;
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```
public class BMISpreadsheetUser {
   public static void main(String[] args) {
      ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
      bmiSpreadsheet.setHeight(1.77);
      bmiSpreadsheet.setWeight(75);
      System.out.println(bmi.getBMI());
   }
}
public class ABMISpreadsheet {
    double height;
    double weight;

    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getBMI() {
        return weight / (height * height);
    }

    public static void main(String[] args) {
        ABMISpreadsheet aBMISpreadsheet = new ABMISpreadsheet(1.77, 75.0);
    }
}
Every Class has a Constructor

```java
public class ABMISpreadsheet {
    double height;
    double weight;

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getBMI() {
        return weight / (height * height);
    }
}
```
public class ABMISpreadsheet {
    double height;
    double weight;
    public ABMISpreadsheet() {
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
A Class Can Have Multiple Constructors

```java
public class ABMISpreadsheet {
    double height;
    double weight;

    public ABMISpreadsheet() {
    }

    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getBMI() {
        return weight/(height*height);
    }
}
```

Overloaded Constructor

Two methods with the same name are overloaded.
The lists of parameter types must be different.
The method headers must be different ignoring the return type.
A Class Can Have Multiple Constructors (Review)

```java
public class ABMISpreadsheet {
    double height;
    double weight;

    public ABMISpreadsheet() {
    }

    public ABMISpreadsheet(double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getWeight() {
        return weight;
    }

    public void setWeight(double newWeight) {
        weight = newWeight;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double newHeight) {
        height = newHeight;
    }

    public double getBMI() {
        return weight/(height*height);
    }
}
```

Overloaded Constructor

Two methods with the same name are overloaded

The lists of parameter types must be different

The method headers must be different ignoring the return type
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        //equivalent computation
        bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
    }
}
**Are (Programmer-Defined) Constructors Ever Absolutely Necessary?**

```java
ABMISpreadsheet aBMISpreadsheet = new ABMISpreadsheet(1.77, 75.0);
```

```java
ABMISpreadsheet aBMISpreadsheet = new ABMISpreadsheet();
aBMISpreadsheet.setHeight(1.77);
aBMISpreadsheet.setWeight(75.0);
```

- **Programmer can initialize state after instantiation (requires a bit more work but possible in this case)**
- **Always possible?**
- **Can use the full functionality of class without programmer-defined constructor**
- **Some part of the exported state (e.g. height) may be readonly**
Immutable Objects

String s = "hello";

String s = new String("hello");

String is immutable.

An immutable object cannot be changed after initialization.

An immutable object with state must have one or more programmer-defined constructors to initialize the state.
### Changing Variable vs. Object

```java
String s = "hello";
String hello = s;
s += " world";
System.out.println(s == hello);
```

**Assigns to s a new String object**

**Does not change the original String**

```java
StringBuffer s = new StringBuffer("hello");
StringBuffer hello = s;
s.append(" world");
System.out.println(s == hello);
```

**Does not reassign s**

**Changes the object to which s points**

**Reassigning a new object less efficient**

false  true
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        double computedBMI = bmiSpreadsheet.getBMI();
        System.out.println(computedBMI);
    }
}

**Objects vs. Primitives**

- **Primitive Variable**: Objects
- **Primitive Value**: Object Value
- **Object Variable**: Object Value
- **Object Value**: Object Value
## Primitives vs. Object Variables

### Primitive Variables

```java
double computedBMI = 22.5;
double weight = 75.8;
```

### Object Variables

```java
ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
```

### Variables vs. Memory

<table>
<thead>
<tr>
<th>Variables</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>computedBMI</td>
<td>22.5</td>
</tr>
<tr>
<td>weight</td>
<td>75.8</td>
</tr>
<tr>
<td>height</td>
<td>1.77</td>
</tr>
<tr>
<td>weight</td>
<td>75</td>
</tr>
<tr>
<td>bmiSpreadsheet</td>
<td>52</td>
</tr>
</tbody>
</table>
PRIMITIVES VS. OBJECTS STORAGE

ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77);
bmiSpreadsheet2 = bmiSpreadsheet1;

<table>
<thead>
<tr>
<th>addresses</th>
<th>variables</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>height</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>0</td>
</tr>
<tr>
<td>64</td>
<td>height</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>0</td>
</tr>
<tr>
<td>bmiSpreadsheet2</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>bmiSpreadsheet1</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>
Uninitialized Primitive Variable

Uninitialized Object Variable

```java
public class BMISpreadsheetUser {
    public static void main(String[] args) {
        ABMISpreadsheet bmiSpreadsheet;
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        double computedBMI;
        System.out.println(computedBMI);
    }
}
```
**Default Values for Variables**

### Primitive Variables
- `double computedBMI;`
- `double weight;`

### Object Variables
- `ABMISpreadsheet bmiSpreadsheet;`

### Variables and Memory
- `computedBMI` in memory: 0.0
- `weight` in memory: 0.0
- `bmiSpreadsheet` in memory: null

**Legal double values**

**Illegal ABMISpreadsheet value**
**Invoking Methods on null**

- `bmiSpreadsheet.getBMI()`
  - null pointer exception
  - Exception is an unexpected event (error)
  - Guilty method will be terminated and exception reported
  - Will see other exceptions later
Extra
**Why Immutable String?**

- Easier to implement (do not have to address insertions)
- Immutable objects make it easier to implement correct programs with threads and hashtables

```java
String s1 = "hello world";
String s2 = "hello world";
System.out.println(s1 == s2);
```

- Allows literals (String constants) to share memory location
- StringBuffer supports mutable strings
**Why Immutable String?**

```java
String s1 = new String ("hello world");
String s2 = new String ("hello world");
System.out.println(s1 == s2);
```

false

New String Allocated

StringBuffer supports mutable strings