Comp 401: Constructors and Pointers

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

A Class (With Properties)

```
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

Weight

Height

BMI

```
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);
  }
}
```

Using BMISpreadsheet

```
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());
    }
}
```

ABMISpreadsheet and Constructor

```
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;
  }
  publi
    ret
  }
}
```

Constructor name must be the name of the class

Constructor name is also the type of object returned

Constructors do not appear in interfaces

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

Every Class has a Constructor

```
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);
  }
}
```

Equivalent Class Created by Java

```
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);
  }
}
```

Inserted in Object Code not in Source Code

If Programmer Specifies no Constructor, Java inserts a null constructor

A Class Can Have Multiple Constructors

```
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)

```
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

Using Overloaded Constructors

```
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);
  }
}
```

Using Overloaded Constructor

Are (Programmer-Defined) Constructors ever Absolutely Necessary?

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

```
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

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

false

Assigns to s a new String object

Does not change the original String

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

true

Does not reassign s

Reassigning a new object less efficient

Changes the object to which s points

Objects vs. Primitives

```
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 );
    }
}
```

Primitive Variable

Primitive Value

Object Variable

Object Value

Primitives vs. Object Variables

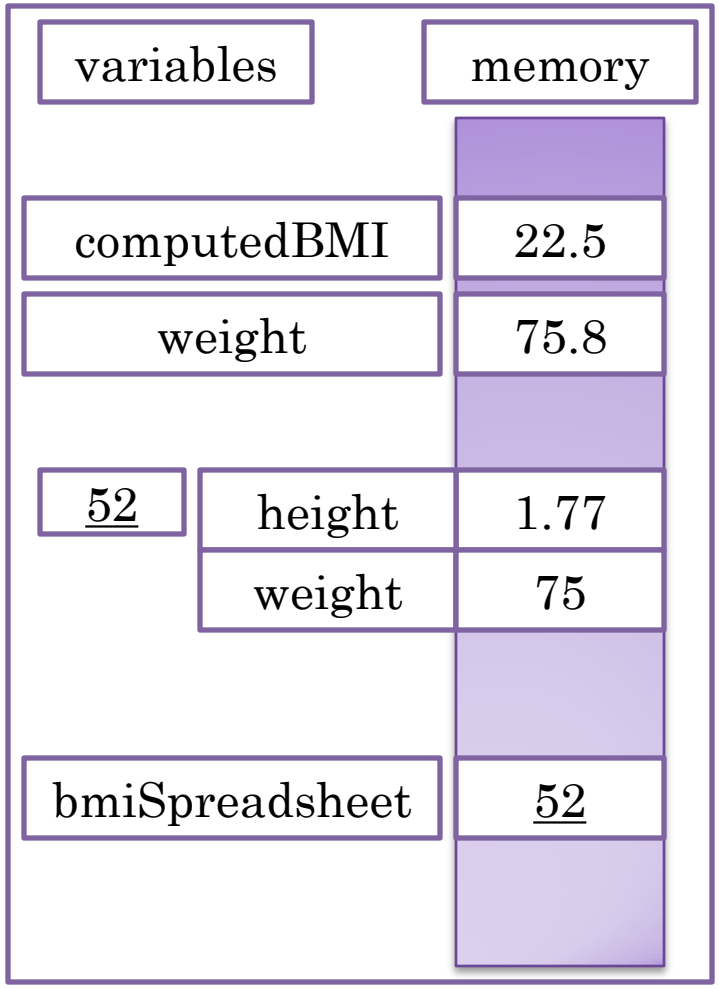
Primitive Variables

double computedBMI = 22.5;

double weight = 75.8;

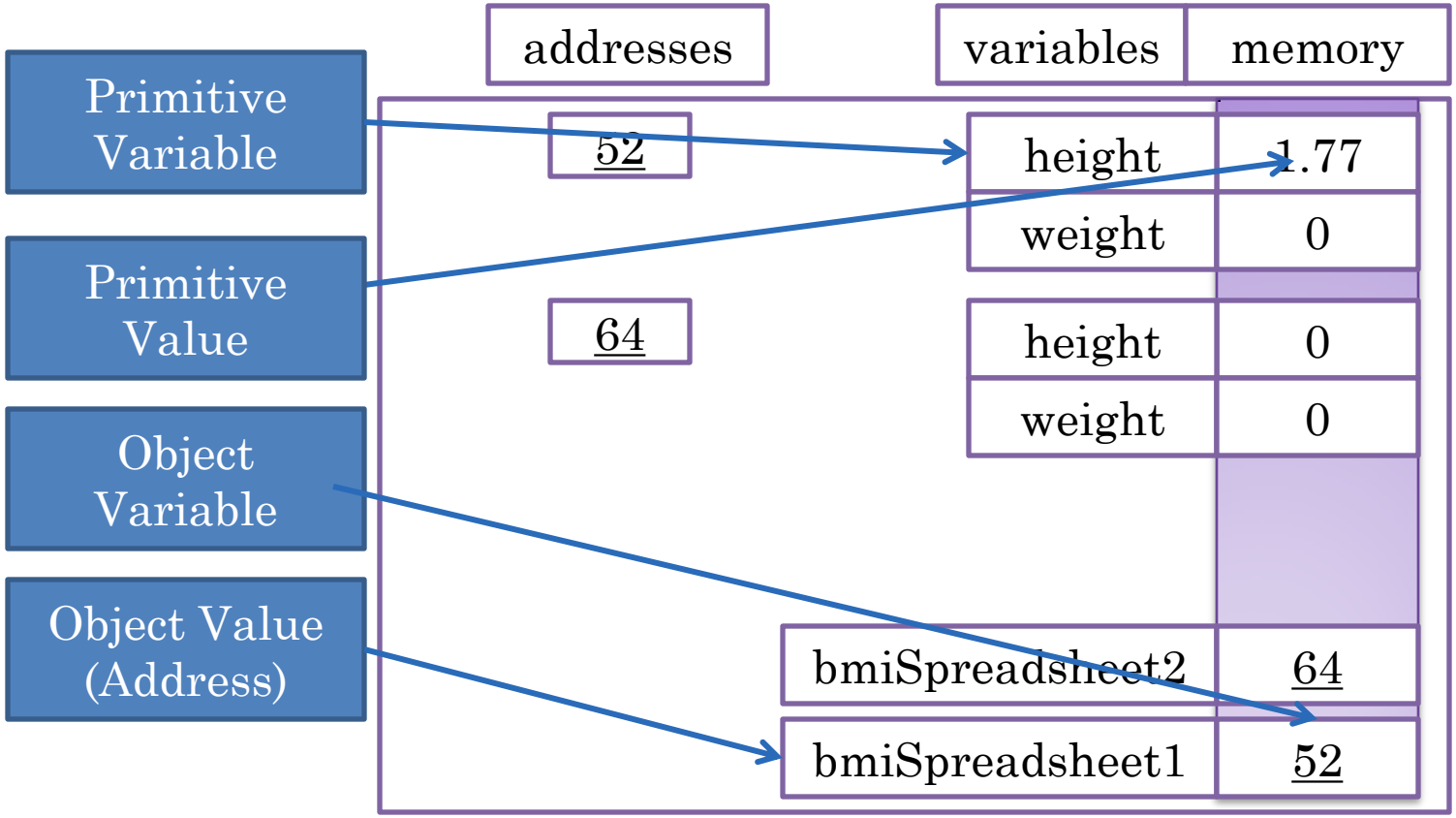
Object Variables

ABMISpreadsheet bmiSpreadsheet = **new** ABMISpreadsheet(1.77, 75);



Primitives vs. Objects Storage

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



Uninitialized Primitive vs. Object Variables

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

Uninitialized Primitive Variable

Uninitialized Object Variable

Default Values for Variables

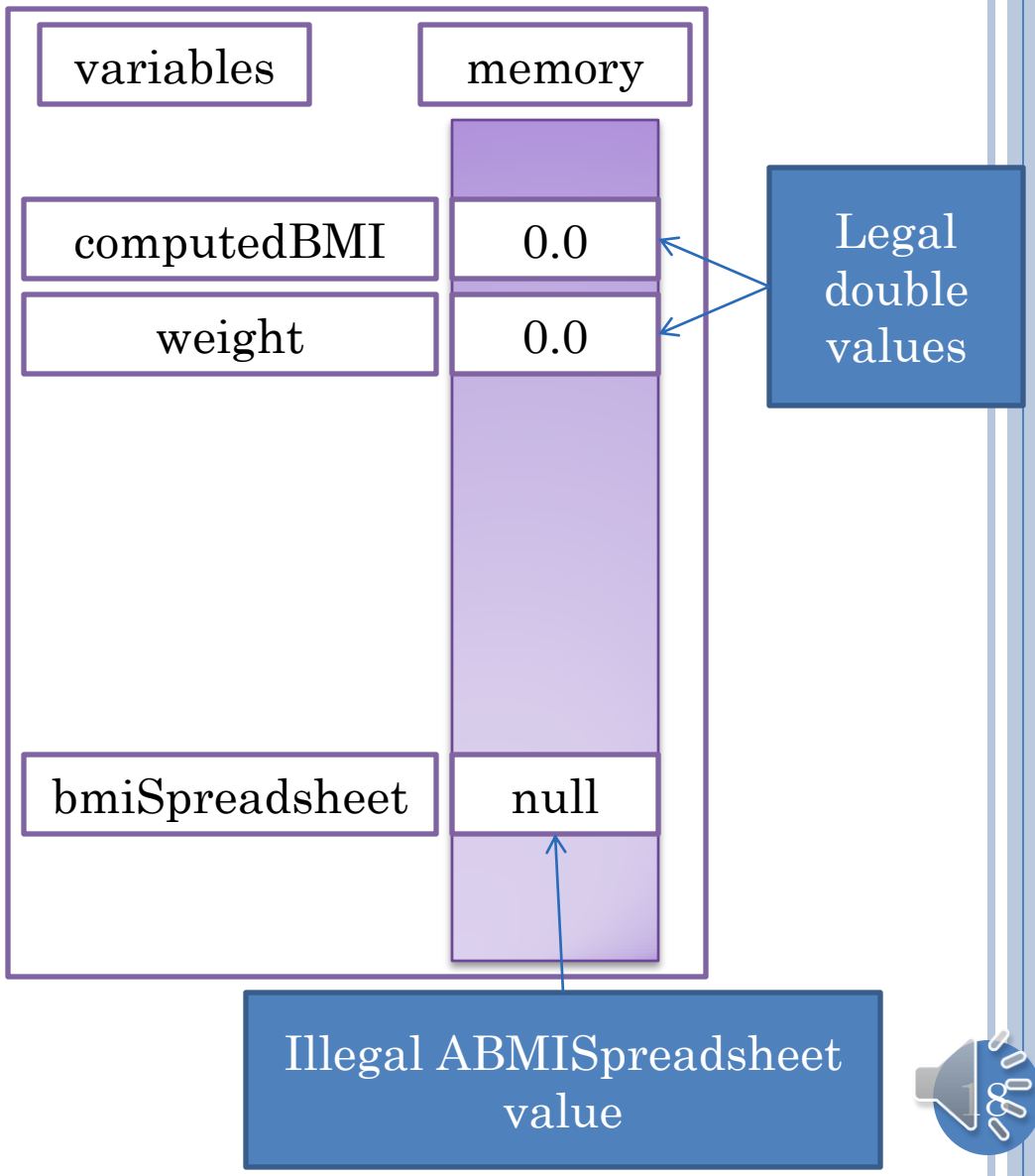
Primitive Variables

double computedBMI;

double weight;

Object Variables

ABMISpreadsheet bmiSpreadsheet;



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 is easier to implement correct programs with threads and hashtables

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

```
true
```

Allows literals (String constants) to share memory location

StringBuffer supports mutable strings

Why Immutable String?

```
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