COMP 401
STATE

Instructor: Prasun Dewan
Prerequisites

- Objects
public class ABMICalculator
{
    public double calculateBMI(double weight, double height)
    {
        return weight / (height * height);
    }
}

public class BMICalculatorTester
{
    public static void main(String[] args) {
        ABMICalculator bmiCalculator = new ABMICalculator();
        System.out.println(bmiCalculator.calculateBMI(75, 1.77));
    }
}
public class ABMICalculator {
    public double calculateBMI(double weight, double height) {
        return weight/(height*height);
    }
}

Each time formal parameters are assigned new actual parameters
“What if” BMI Calculations With General Purpose Calculator

Must re-enter height each time!
"What if" BMI Calculations With Specialized Calculator

```java
public double calculateMyBMI(double weight) {
    final double MY_HEIGHT = 1.77;
    return (new ABMICalculator).calculateBMI(weight, MY_HEIGHT);
}
```

Must only enter the weight

But the height is hardwired! Must create a separate class for each user!

General purpose solution that does not require re-entry of height each time?
Calculate two BMIs using one instance of ABMISpreadsheet and changing only the weight

State: Data remembered by an object between method invocations
**Instance Variables**

ABMICalculator Instance

- calculateBMI
  - Body
  - Parameters
  - Belong to a single method
  - Local variable

ABMISpreadsheet Instance

- getBMI
  - Body
  - Instance Variables
  - accesses
  - Belong to all methods of an instance
  - Global variable
**State-less vs. State-full Objects**

**Identical Instances ~ car radios with no presets**

**Different Instances ~ car radios with presets**
public class ABMISpreadsheet {
    double height;
    ...
    double weight;
    ...
    public double getBMI() {
        return weight/(height*height);
    }
    ...
}
public class ABMISpreadsheet {
    double height;
    ...
    double weight;
    ...
    public double getBMI() {
        return weight/(height*height);
    }
    ...
}
ACCESSING INSTANCE VARIABLES VIA PUBLIC METHODS

ABMISpreadsheet Instance

weight
-reads
-getWeight()
calls
-weight

setWeight()
calls

write

height
-reads
-getHeight()
calls

write

setHeight()
calls

getBMI()
calls

ObjectEditor

new weight

new height

height


Coding Getter and Setter Methods

ABMISpreadsheet Instance

weight

getWeight()  setWeight()

reads  writes

calls  calls

weight  new weight

ObjectEditor
Coding Getter and Setter Methods

ABMISpreadsheet Instance

- weight
  - gets weight
  - sets new weight

ObjectEditor

- calls
  - writes

public double getWeight()
{
    return weight;
}

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

procedure – returns nothing

function
public class ABMISpreadsheet {
    double height;

    public double getHeight() {
        return height;
    }

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

    double weight;

    public double getWeight() {
        return weight;
    }

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

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

FUNCTION vs. PROCEDURE

function

procedure – returns nothing
Function vs. Procedure

- **procedure:** deposit
- **function:** withdraw
**Pure vs. Impure Functions**

**ABMICalculator Instance**
- `calculateBMI`
- `Body`
  - `weight`
  - `height`
- `getWeight` accesses `weight`
- `setWeight(77)`
- `calculateBMI(77, 1.77) 24.57`
- `...`
- `calculateBMI(77, 1.77) 24.57`

**ABMISpreadsheet Instance**
- `getWeight`
- `Body`
  - `weight`
- `setWeight(77)`
- `getWeight()` accesses `weight`
- `getWeight()`
- `setWeight(71)`
- `getWeight()`
- `71`
Functions with Side Effects

```java
public class ASquareAndCubeSpreadsheetWithSideEffects {
    int number;
    int square;

    public void setNumber(int theNumber) {
        number = theNumber;
    }

    public int getNumber() {
        return number;
    }

    public int getSquare() {
        square = number * number;
        return square;
    }

    public int getCube() {
        int retVal = square * number;
        System.out.println("The Cube is: " + retVal);
        return retVal;
    }
}
```
Functions with Side Effects

```java
public class ASquareAndCubeSpreadsheetWithSideEffects {
    int number;
    int square;

    public void setNumber(int theNumber) {
        number = theNumber;
    }

    public int getNumber() {
        return number;
    }

    public int getSquare() {
        square = number*number;
        return square;
    }

    public int getCube() {
        int retVal = square*number;
        System.out.println("The Cube is: " + retVal);
        return retVal;
    }
}
```

Side effect: Changing global state or printing (non debugging) output in function

Unexpected: makes function behave like a procedure

Side effects are confusing and should be avoided in the functions you write in this course.
**FUNCTION”NESS” OF METHOD**

- **Pure Function**: computes a value
  - Does not access global variables or produce (non-debug) output

- **Impure Function without side effects**: computes a value
  - Does not write global variables

- **Impure Function with side effects**: computes a value
  - Reads/writes global variables or produces (non-debug) output

- **Procedure**: returns nothing
  - Reads/writes global variables/produces output
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
**Read-only and Editable Properties**

Typed, Named Unit of Exported Object State

```java
public class C {

    public T getP() {
        ...
    }

    public void setP(T newValue) {
        ...
    }
}
```

- **Bean convention:** For humans and tools
- **Name P**
- **Type T**
- **Read-only**
- **Editable**
- **Getter method**
- **Setter method**

Violates Bean convention: obtainP newP
**READ-ONLY AND EDITABLE PROPERTIES (review)**

Typed, Named Unit of Exported Object State

```java
public class C {
    
    public T getP() {
        ...
    }

    public void setP(T newValue) {
        ...
    }

    Bean
    Name P
    Type T
    Read-only
    Editable
    Getter method
    Setter method

    newP

    Bean convention: For humans and tools

    obtainP

    Violates Bean convention
```
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        System.out.println("getHeight Called");
        return height;
    }
    public void setHeight(double newHeight) {
        System.out.println("setHeight Called");
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        System.out.println("getWeight Called");
        return weight;
    }
    public void setWeight(double newWeight) {
        System.out.println("setWeight Called");
        weight = newWeight;
    }
    public double getBMI() {
        System.out.println("getBMI Called");
        return weight/(height*height);
    }
}
Actual Trace

Extra `getWeight()` call made by the undo-redo mechanism in `ObjectEditor`
DISPLAYING AND THEN CHANGING OBJECT

```java
public class ABMISpreadsheetManipulatedByMainAndObjectEditor {
    public static void main (String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        ObjectEditor.edit(bmiSpreadsheet);
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
    }
}
```

Setters not called through ObjectEditor, so it does not know it should refresh
REFRESHING OBJECTEDITOR FROM MAIN

```java
public class ABMISpreadsheetRefreshedByMain {
    public static void main (String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        OEFrame oeFrame = ObjectEditor.edit(bmiSpreadsheet);
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        oeFrame.refresh();
    }
}
```

Better ways to refresh we will learn later.
public class ABMISpreadsheetAnimatingDemoer {
    public static void main (String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
        OEReFrame editor = ObjectEditor.edit(bmiSpreadsheet);
        ThreadSupport.sleep(5000);
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(70);
        editor.refresh();
        ThreadSupport.sleep(5000);
        editor.select(bmiSpreadsheet, "Height");
        bmiSpreadsheet.setHeight(0);
        editor.refresh();
        ThreadSupport.sleep(5000);
        editor.select(bmiSpreadsheet, "Weight");
        bmiSpreadsheet.setWeight(0);
        editor.refresh();
    }
}

ThreadSupport() makes program wait for specified number of milliseconds
DEMO

https://www.youtube.com/watch?v=dYfSuP3Io8I&feature=plcp
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class ABMISpreadsheet {
    double hght;  
    public double getHowTall() {  
        return hght;  
    }  
    public void setHowTall(double newHeight) {  
        hght = newHeight;  
    }  
    double weight;  
    public double getWeight() {  
        return weight;  
    }  
    public void setWeight(double newWeight) {  
        weight = newWeight;  
    }  
    public double getBMI() {  
        return weight/(hght*hght);  
    }  
}

HowTall
Weight
BMI
public class ABMICalculator {
    public double calculateBMI (double weight, double height) {
        return weight/ (height * height);
    }
}

No Properties
NO_PATTERN Annotation

import util.annotations.StructurePattern
@StructurePattern(StructurePatternNames.NO_PATTERN)
public class ABMICalculator {
    public double calculateBMI (double weight, double height) {
        return weight / (height * height);
    }
}

Annotation is like a comment except it is typed and available at runtime

Available to ObjectEditor

Structure(<PatternName>) before class asserts that the class is following the pattern.
import util.annotations.StructurePattern
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {
        return weight;
    }
    public void set(double newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(int newHeight) {
        height = newHeight;
    }
    public double BMI() {
        return weight/(height*height);
    }
}
(Editable) Property Name Annotations

```java
import util.annotationsEditable.PropertyNames;
import util.annotations.PropertyNames;

@StructurePattern (StructurePatternNames.BEAN_PATTERN)
@PropertyNames([{ "Height", "Weight", "BMI" }])
@EditablePropertyNames({"Height", "Weight"})

public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {
        return weight;
    }
    public void set(double newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
    public double BMI() {
        return weight/(height*height);
    }
}
```
Order of Properties

@StructurePattern(StructurePatternNames.BEAN_PATTERN)
@PropertyNames({"Weight", "Height", "BMI"})
@EditablePropertyNames({"Height", "Weight"})

public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {
        return weight;
    }
    public void set(double newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(int newHeight) {
        height = newHeight;
    }
    public double BMI() {
        return weight/(height*height);
    }
}
**OVERLOADING**

Look at that plane fly.

The fly is bothering me.

Two different words with the same name

Operation Definitions

Context of actual parameters

Two different operations with the same name

```java
public void println(String val) {...}  // System.out.println("setWeight called");
public void println(double val) {...}  // System.out.println(newWeight);
```

Two different words with the same name
MORE ON PRINTLN

```java
System.out.println("setWeight called");
System.out.println(newWeight);
System.out.println("setWeight called" + newWeight);
```

Operator Overloading

5 + 6
**Ambiguous Context**

Time flies like an arrow.

Fruit flies like an orange.

```java
public void println(String val) {...}
System.out.println("setWeight called");
```

```java
public void println(String val) {...}
System.out.println(newWeight);
```

- **Operation Definitions**
- **Java cannot use context to disambiguate**

Defining two versions of a method?  
Why is overloading useful?
INCONSISTENT BMI STATE

ObjectEditor.edit(new ABMISpreadsheet());
Fixing Inconsistent BMI State

ABMISpreadsheet aBMISpreadsheet = new ABMISpreadsheet();
aBMISpreadsheet.setHeight(1.77);
aBMISpreadsheet.setWeight(75.0);
Always Consistent BMI State

ABMISpreadsheet aBMISpreadsheet = new ABMISpreadsheet(1.77, 75.0);
public class ABMISpreadsheet {
    double height, weight;
    public ABMISpreadsheet(
        double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getHeight() {
        return height;
    }

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

    public double getWeight() {
        return weight;
    }

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

    public double getBMI() {
        return weight/(height*height);
    }
}
public class ABMISpreadsheet {
    double height, weight;
    public ABMISpreadsheet(
        double theInitialHeight, double theInitialWeight) {
        setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }

    public double getHeight() {
        return height;
    }

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

    public double getWeight() {
        return weight;
    }

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

    public double getBMI() {
        return weight/(height*height);
    }
}
Every Class has a Constructor

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

    public double getHeight() {
        return height;
    }

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

    public double getWeight() {
        return weight;
    }

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

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

  public ABMISpreadsheet() {}  // Inserted in Object Code not in Source Code

  public double getHeight() {
    return height;
  }

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

  public double getWeight() {
    return weight;
  }

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

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

If Programmer Specifies no Constructor, Java inserts a null constructor
A Class Can Have Multiple Constructors

```java
public class ABMISpreadsheet {
    double height, weight;
    public ABMISpreadsheet() {}
    public ABMISpreadsheet(
        double theInitialHeight, double theInitialWeight) {
       .setHeight(theInitialHeight);
        setWeight(theInitialWeight);
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    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());
        // equivalent computation
        bmiSpreadsheet = new ABMISpreadsheet(1.77, 75);
        System.out.println(bmi.getBMI());
    }
}
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 = 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.
ASSIGNING TO STRING VARIABLE

s = s + " world";

Assigns to s a new String object

Does not change the original String
### 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

```java
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
CHANGING VARIABLE VS. OBJECT

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

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

Does not reassign sb

Reassigning a new object less efficient

Changes the object to which sb points
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);
    }
}
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);
    }
}
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**

```java
double computedBMI;
double weight;
```

**Object Variables**

```java
ABMISpreadsheet bmiSpreadsheet;
```

![Diagram showing default values for variables:]

- variables
  - computedBMI: 0.0
  - weight: 0.0
- memory
  - bmiSpreadsheet: 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
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77)
bmiSpreadsheet2 = bmiSpreadsheet1;

Stack: area at bottom of memory where slots for local variables (formal parameters, internal variables get allocated, stack grows upwards

Heap: area at top of memory where objects get allocated, heap grows downwards

bmiSpreadsheet1 and bmiSpreadsheet2 are internal variables of some method

Stack: area at bottom of memory where slots for local variables (formal parameters, internal variables get allocated, stack grows upwards

Heap: area at top of memory where objects get allocated, heap grows downwards

memory

bmiSpreadsheet2 null
bmiSpreadsheet1 null
New Instance Created

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

New instance created from heap

<table>
<thead>
<tr>
<th>addresses</th>
<th>variables</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>height</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>0</td>
</tr>
<tr>
<td>bmiSpreadsheet2</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>bmiSpreadsheet1</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77);
bmiSpreadsheet2 = bmiSpreadsheet1;

Address of new instance assigned bmiSpreadsheet1

<table>
<thead>
<tr>
<th>addresses</th>
<th>variables</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>height</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet2</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet1</td>
<td>52</td>
</tr>
</tbody>
</table>

Heap

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

New instance created from heap

New Instance Created
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77);
bmiSpreadsheet2 = bmiSpreadsheet1;

Address of second instance assigned bmiSpreadsheet2
**START OF setHeightCall**

```java
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>0</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></td>
<td>newHeight</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet2</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet1</td>
<td>52</td>
</tr>
</tbody>
</table>

Formal parameter of setHeight allocated from stack
After setHeight Assignment

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

Formal parameter of setHeight assigned to instance variable
**setHeight Returns**

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

```
newHeight popped from stack
```

```
<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></td>
<td>bmiSpreadsheet2</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet1</td>
<td>52</td>
</tr>
</tbody>
</table>
```
After Assignment

```java
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></td>
<td>height 1.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weight 0</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>height 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>weight 0</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>bmiSpreadsheet2 52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bmiSpreadsheet1 52</td>
<td></td>
</tr>
</tbody>
</table>

== does an address (pointer) copy
**AFTER GARBAGE COLLECTION**

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>bmiSpreadsheet2</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>bmiSpreadsheet1</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

Unreferenced object garbage collected later
ASSIGNMENT IS NOT A COPY

bmiSpreadsheet2 = bmiSpreadsheet1;
bmiSpreadsheet1.setWeight(75);

bmiSpreadsheet1.getWeight() == bmiSpreadsheet1.getWeight() → true
```
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77);
bmiSpreadsheet2 = bmiSpreadsheet1;
```
Functions with Side Effects

After calling `setNumber()`, ObjectEditor calls `getCube()` before calling `getSquare()` to do automatic refresh.

Explicit refresh calls getters again, and `square` has the correct value now.
EXTRA SLIDES
WHY IMMUtable STRING

Easier to implement (do not have to move string)

Do not have to create a physical copy of a string

```java
s1 = "world";
s2 = s1;
s.setCharAt(1, '0');
System.out.println(s2.charAt(1));
```

Make is easier to implement correct programs with threads and hashtables
CLASSIFYING METHODS

- Procedures
  - return nothing
  - write global variables and produce output

- Functions
  - Return values.
  - Can also write global variables and produce output

- Pure Functions
  - Do not read or write global variables or produce output

- Impure functions:
  - Access global variable or produce output.

- Impure functions without side effects
  - Read global variables but do not write global variables or produce output

- Impure functions with side effects
  - Impure functions write global variables and/or produce output
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {
        return weight;
    }
    public void set(double newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
    public double getHeight() {
        return height;
    }
    public double BMI() {
        return weight / (height * height);
    }
}

Warning if (editable) properties not declared?

Overhead, chances of mistake low, C# has built in support for properties

Why warning if no structure annotation?
(Pattern) Annotation

```java
import util.annotations.StructurePattern;
import util.annotations.StructurePatternNames;
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getWeight() {
        return weight;
    }
    public void set(double newWeight, double newHeight) {
        weight = newWeight;
        height = newHeight;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(int newHeight) {
        height = newHeight;
    }
    public double BMI() {
        return weight / (height * height);
    }
}
```

Annotation is like a comment except it is typed and available at runtime.

Structure(<PatternName>) before class asserts that the class is following the pattern.
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}