Comp 401
State

Instructor: Prasun Dewan

Prerequisites

- Objects

ABMICalculator

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

Instance Method)

Formal Parameters

ABMICalculator type

A variable that holds instances of ABMICalculator

Actual Parameters

```
public class BMICalculatorTester
{
   public static void main (String[] args) {
        ABMICalculator bmiCalculator = new ABMICalculator();
        System.out.println (bmiCalculator.calculateBMI(75, 1.77));
   }
}
```

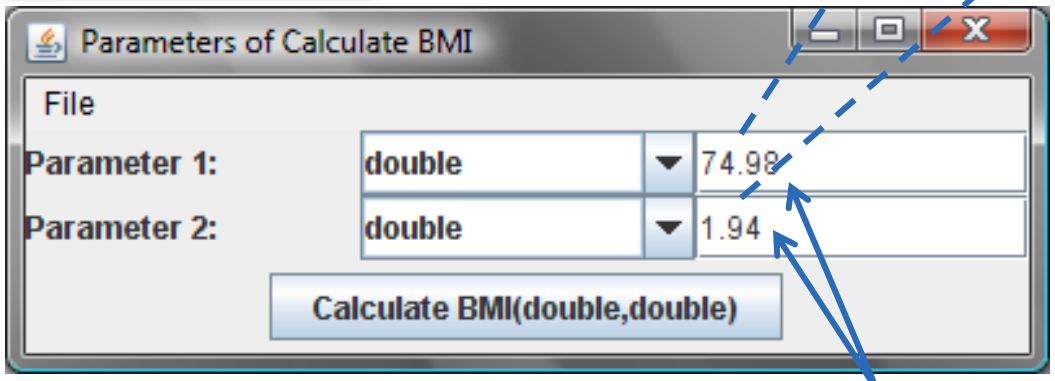
ABMICalculator

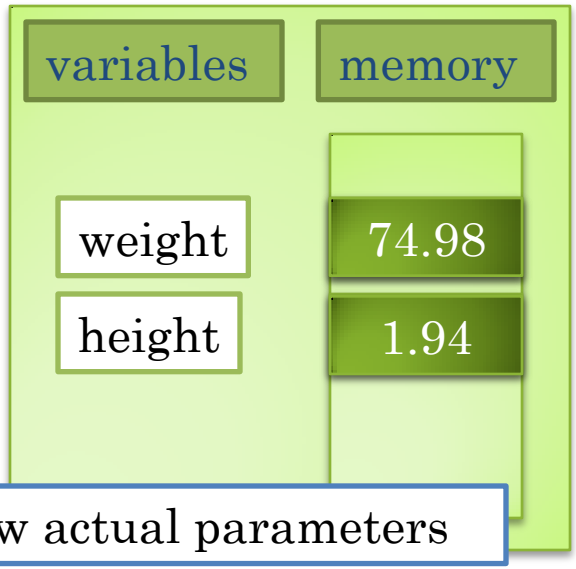
```
public class ABMICalculator
{
    public double calculateBMI(double weight, double height)
    {
        return weight/(height*height);
    }
}
```

Formal Parameters

Invoke calculateBMI

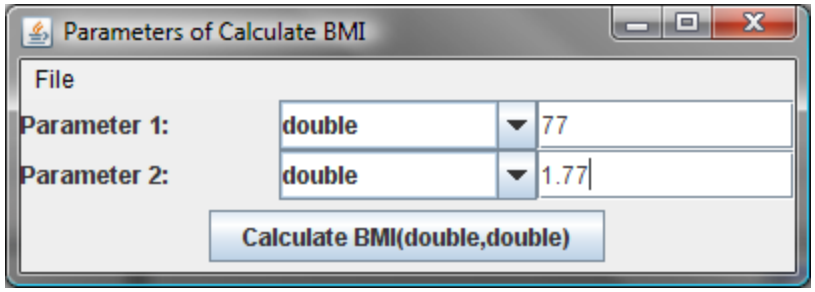
assigned

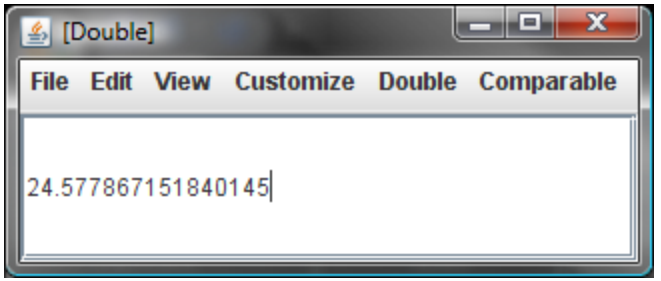


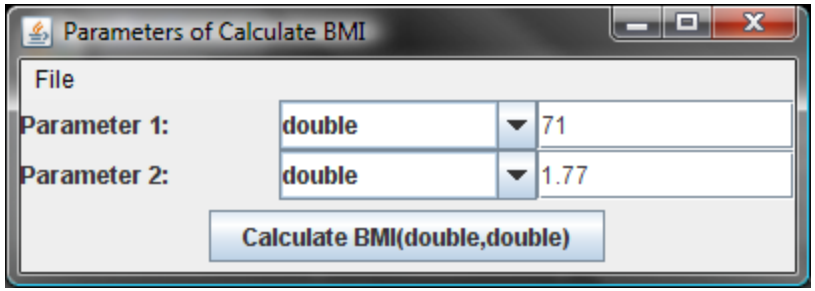


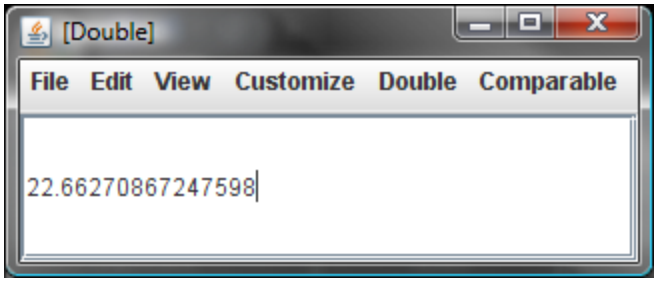
Each time formal parameters are assigned new actual parameters

"What if" BMI Calculations With General Purpose Calculator



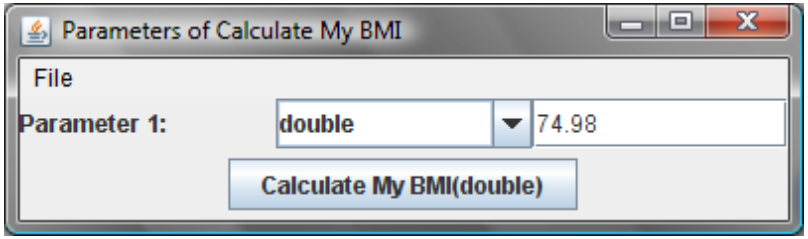






Must re-enter height each time!

"What if" BMI Calculations With Specialized Calculator



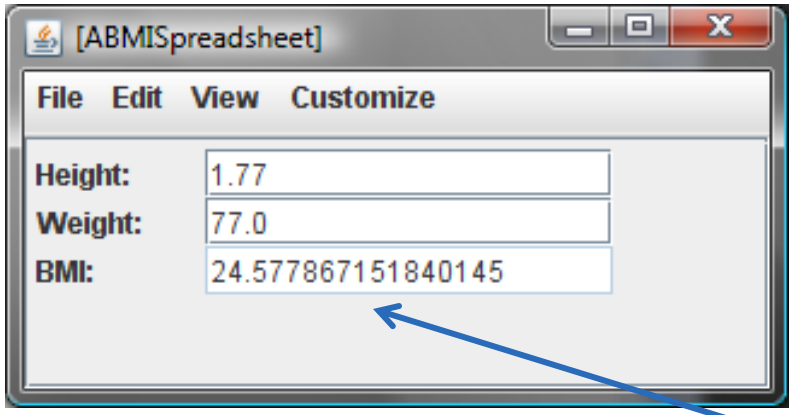

```
public double calculateMyBMI(double weight) {
    final double MY_HEIGHT = 1.77;
    return (new ABMICalculator).calculateBMI(weight, MY_HEIGHT);
}
```

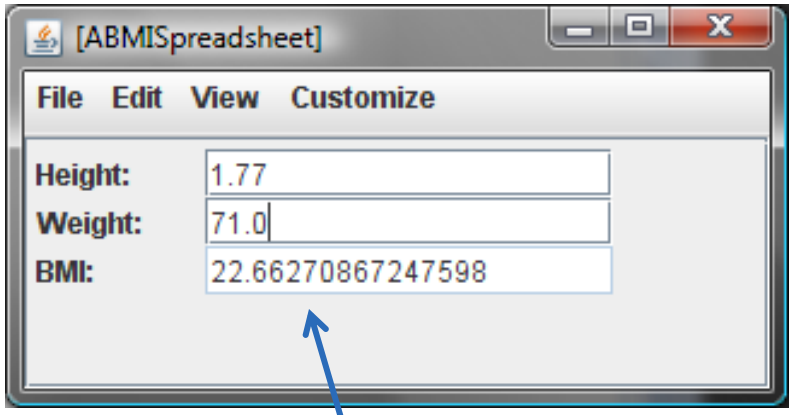
Must only enter the weight

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?

BMI Spreadsheet

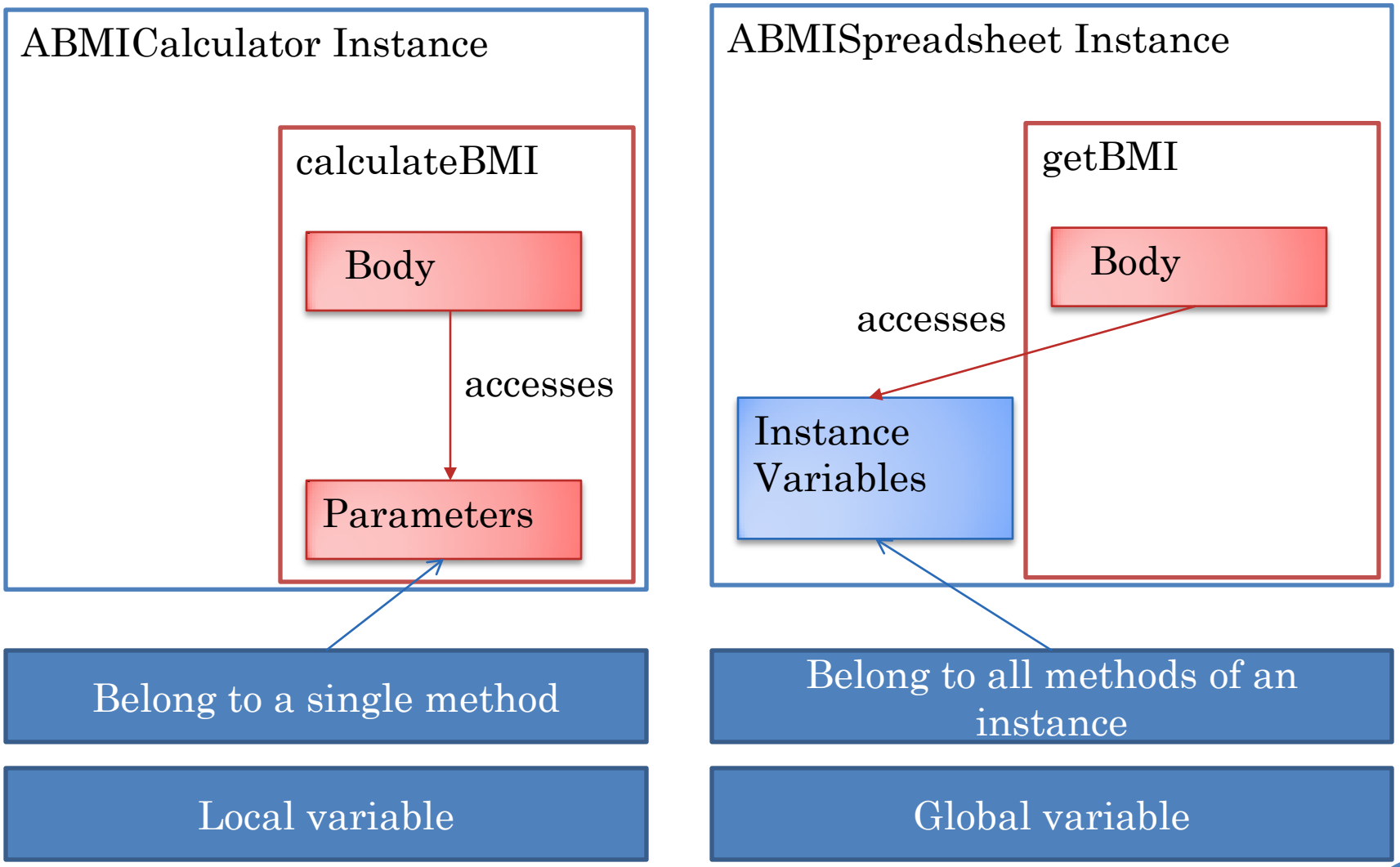




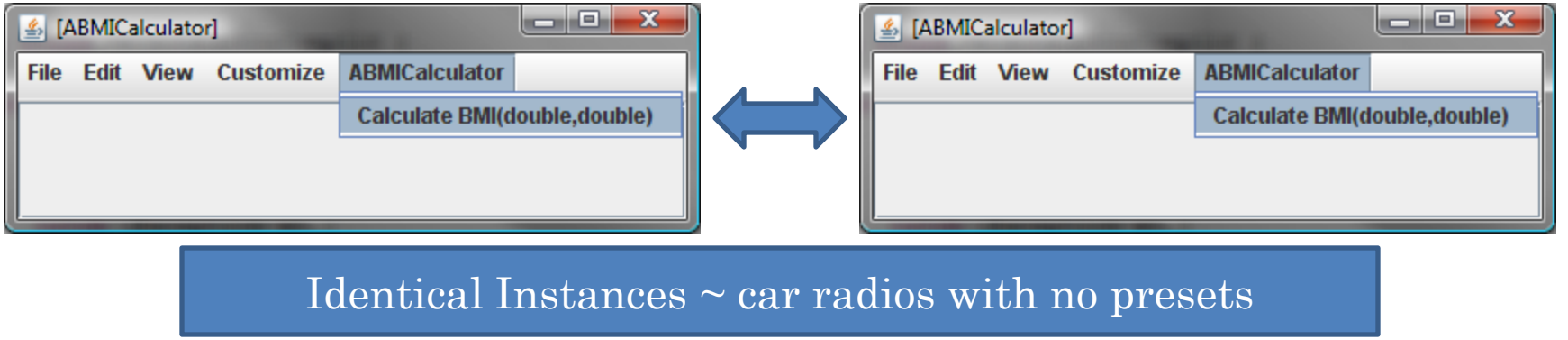
Calculate two BMIs using one instance of ABMISpreadsheet and changing only the weight

State: Data remembered by an object between method invocations

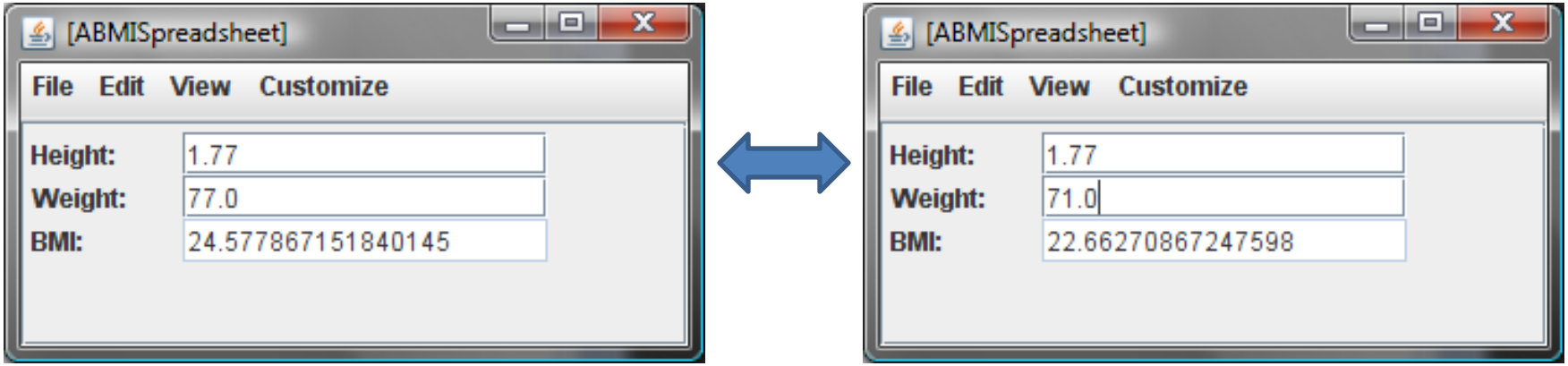
Instance Variables



State-less vs. State-full Objects

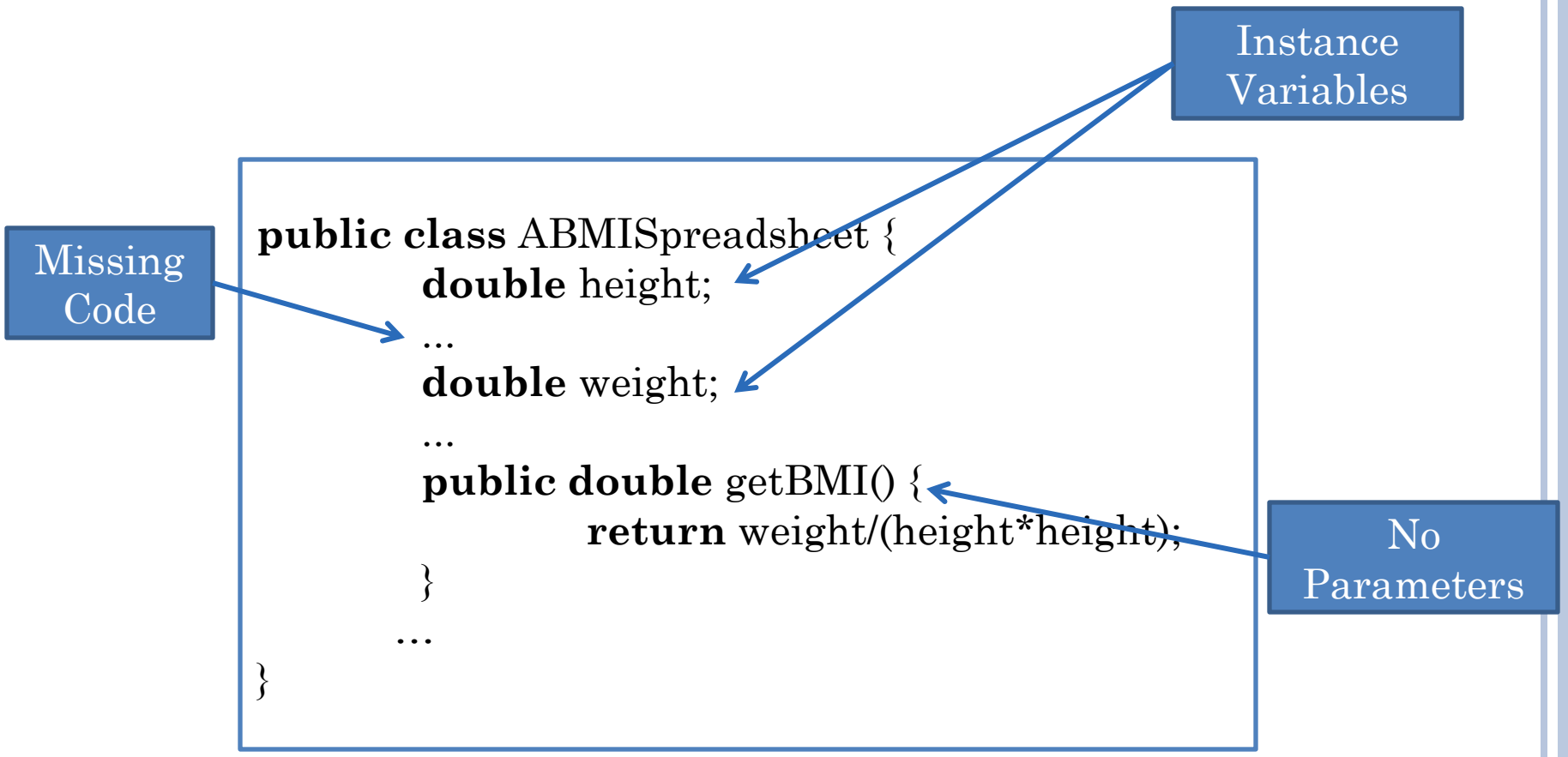


Identical Instances ~ car radios with no presets

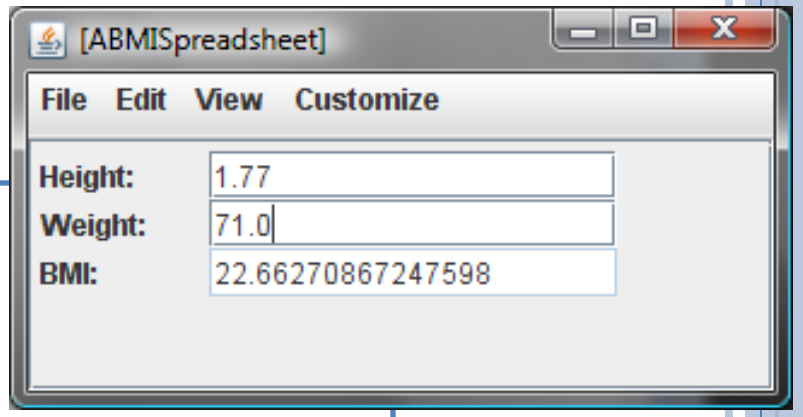


Different Instances ~ car radios with presets

Declaring Instance Variables



Exporting State



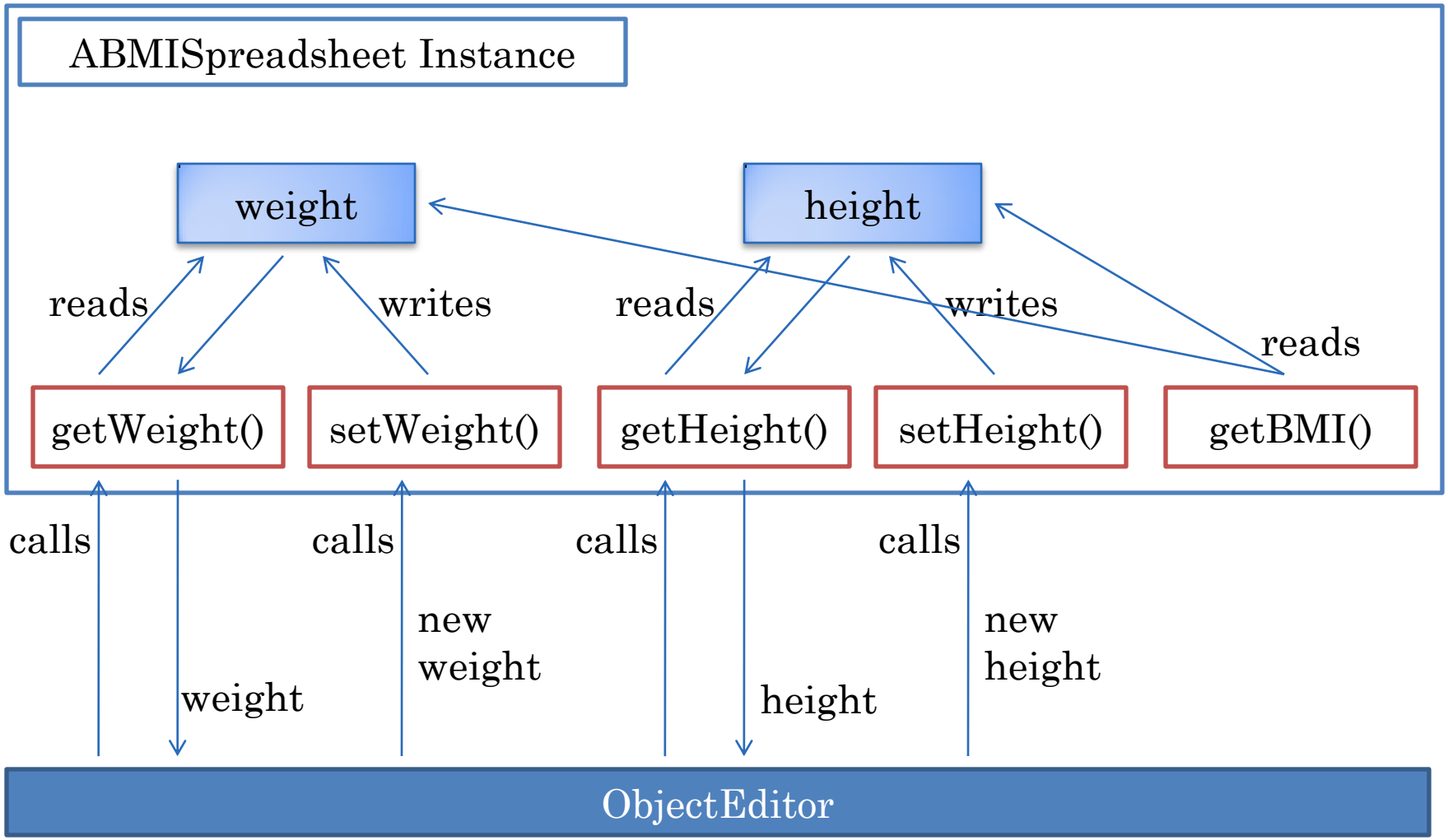

```
public class ABMISpreadsheet {
        double height;
        ...
        double weight;
        ...
        public double getBMI() {
                return weight/(height*height);
        }
      ...
}
```

Outside Access: ObjectEditor

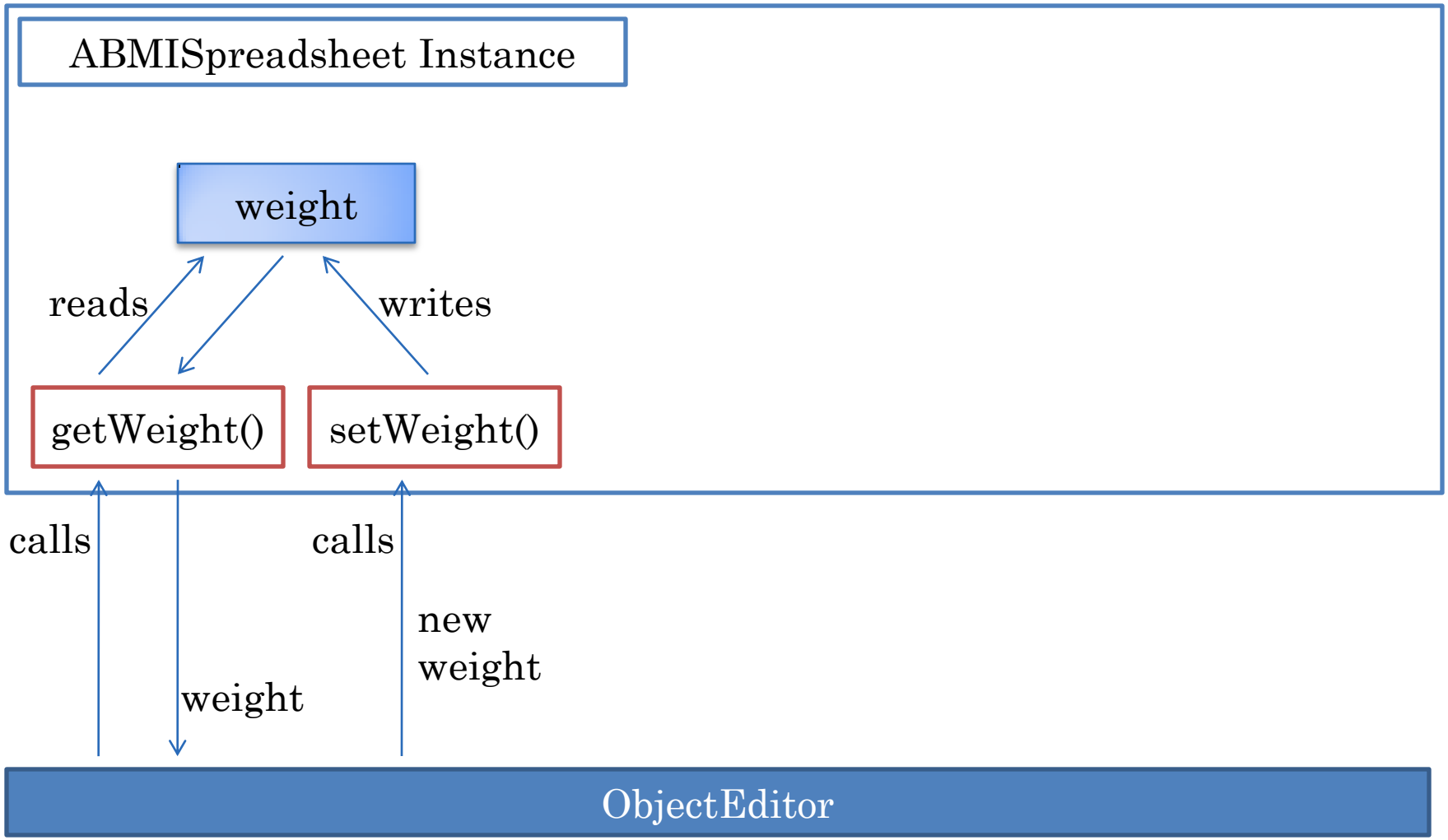
Variables should not be public (like hidden thoughts)

But ObjectEditor needs their values

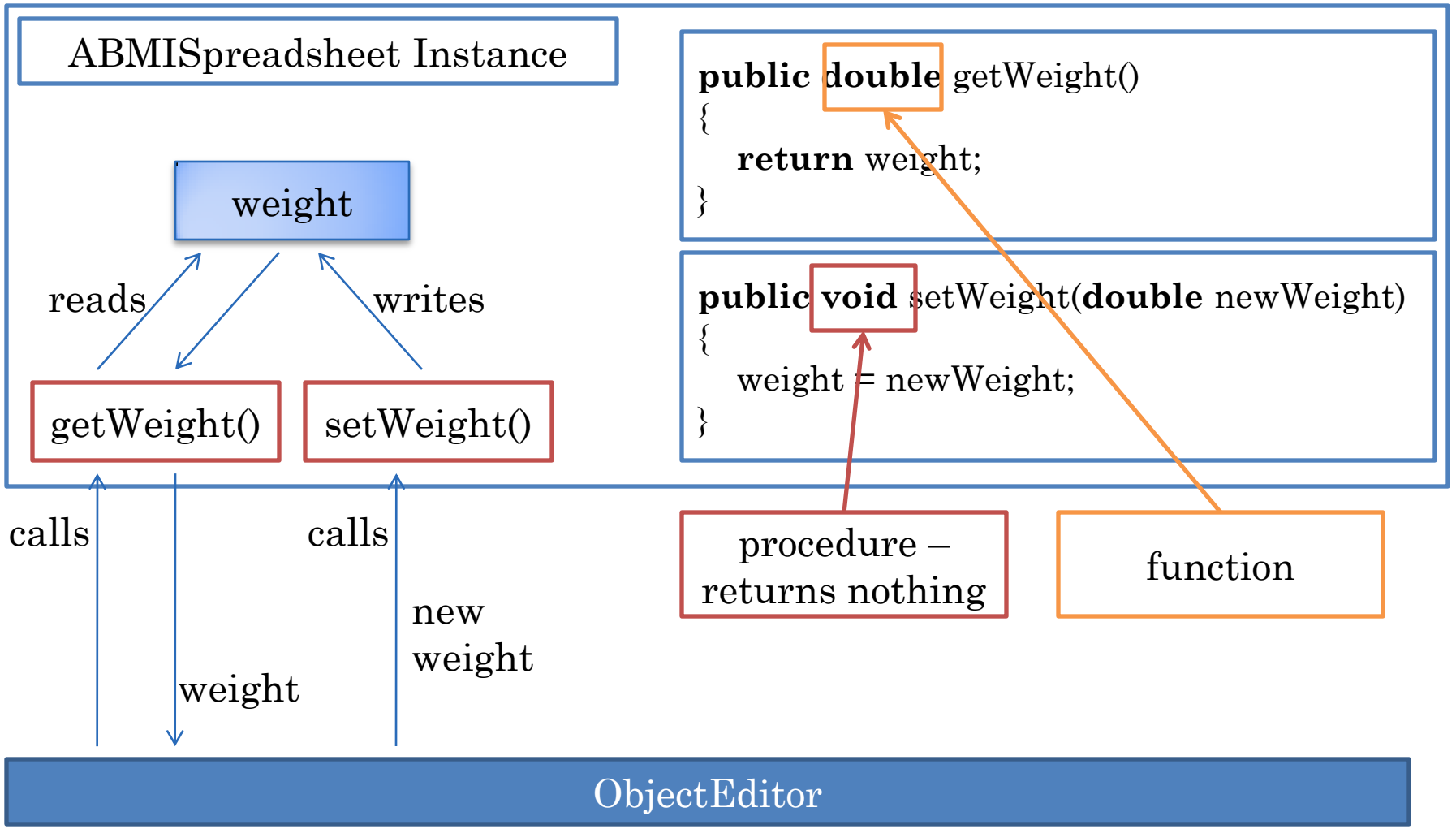
Accessing Instance Variables via Public Methods



Coding Getter and Setter Methods



Coding Getter and Setter Methods



Function vs. Procedure

```
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

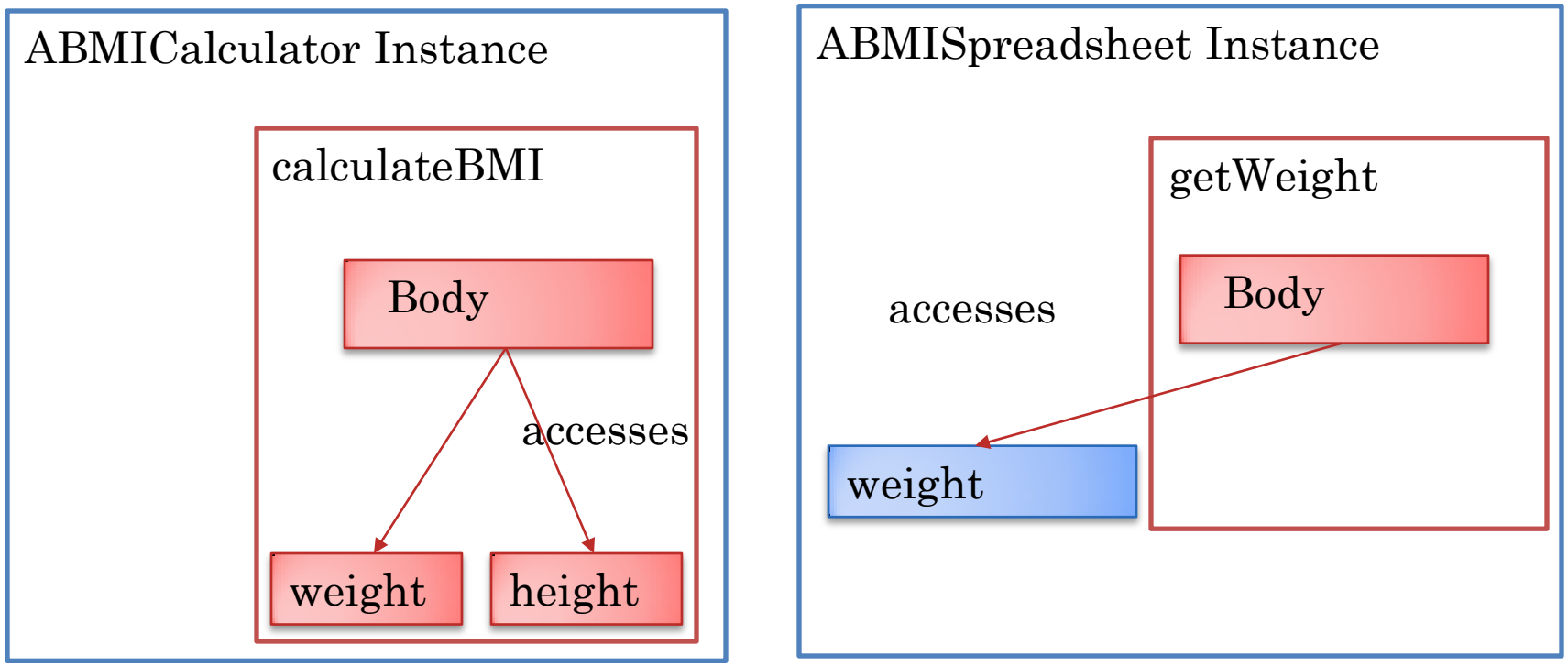
procedure – returns nothing

Function vs. Procedure

procedure:
deposit

function:
withdraw

Pure vs. Impure Functions



calculateBMI(77,1.77) 24.57

...

calculateBMI(77,1.77) 24.57

setWeight(77)

getWeight() 77

setWeight(71)

getWeight() 71

Functions with Side Effects

```
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;
  }
}
```

setNumber(5)	
getCube()	0
getSquare()	25
getCube()	125

Functions with Side Effects

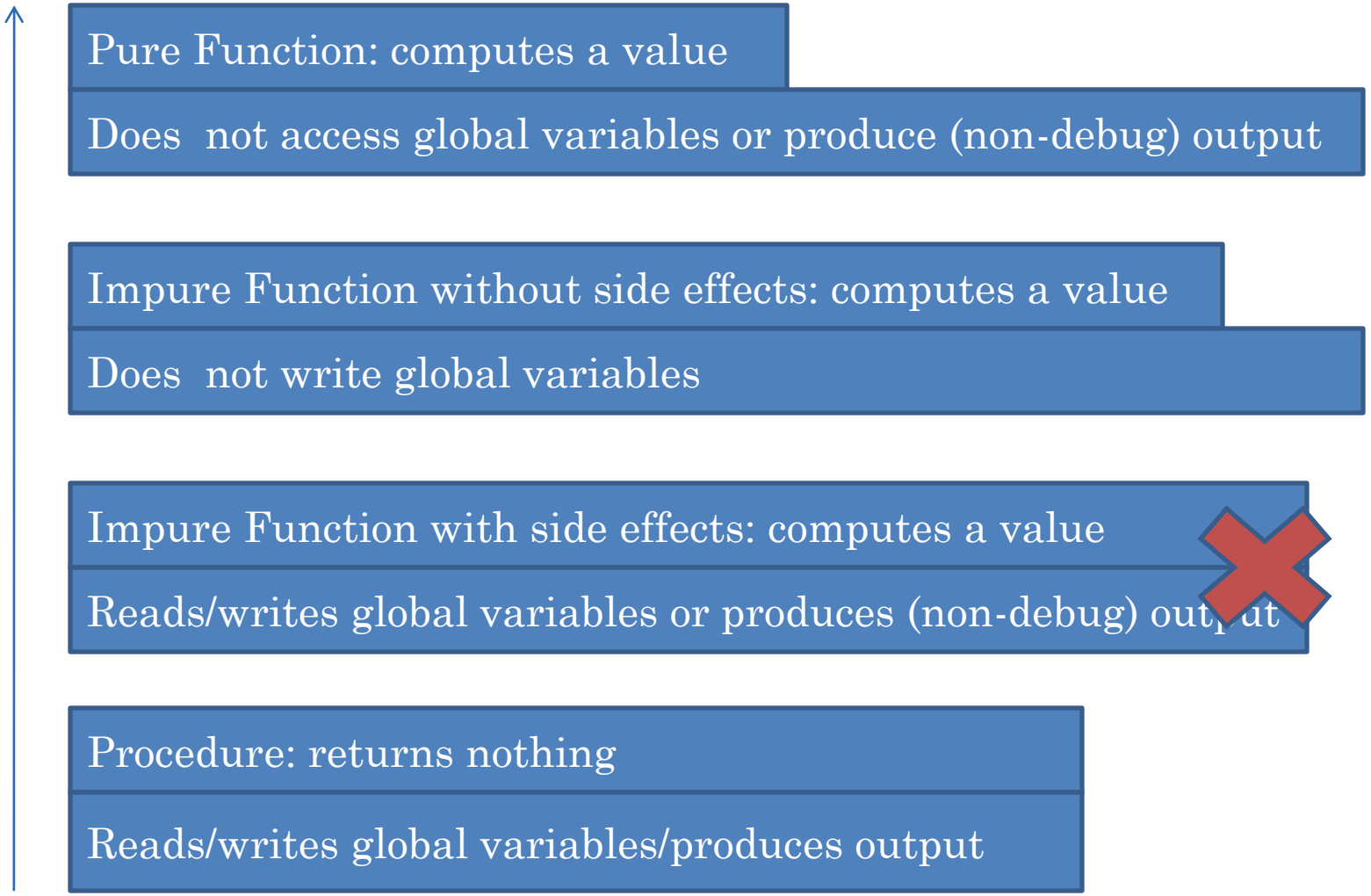
```
public class ASquareAndCubeSpreadsheetWithSideEffects {
   int number;
   int square;
   public void setNumber(int theNumber) {
      number = theNumber;
   }
   public int getNumber() {
      return number;
   }
   public int getSquare() {
      square = number*number;
      return square;
   }
   public int getCube() {
      int retVal = square*number;
      System.out.println("The Cube is: " +
      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

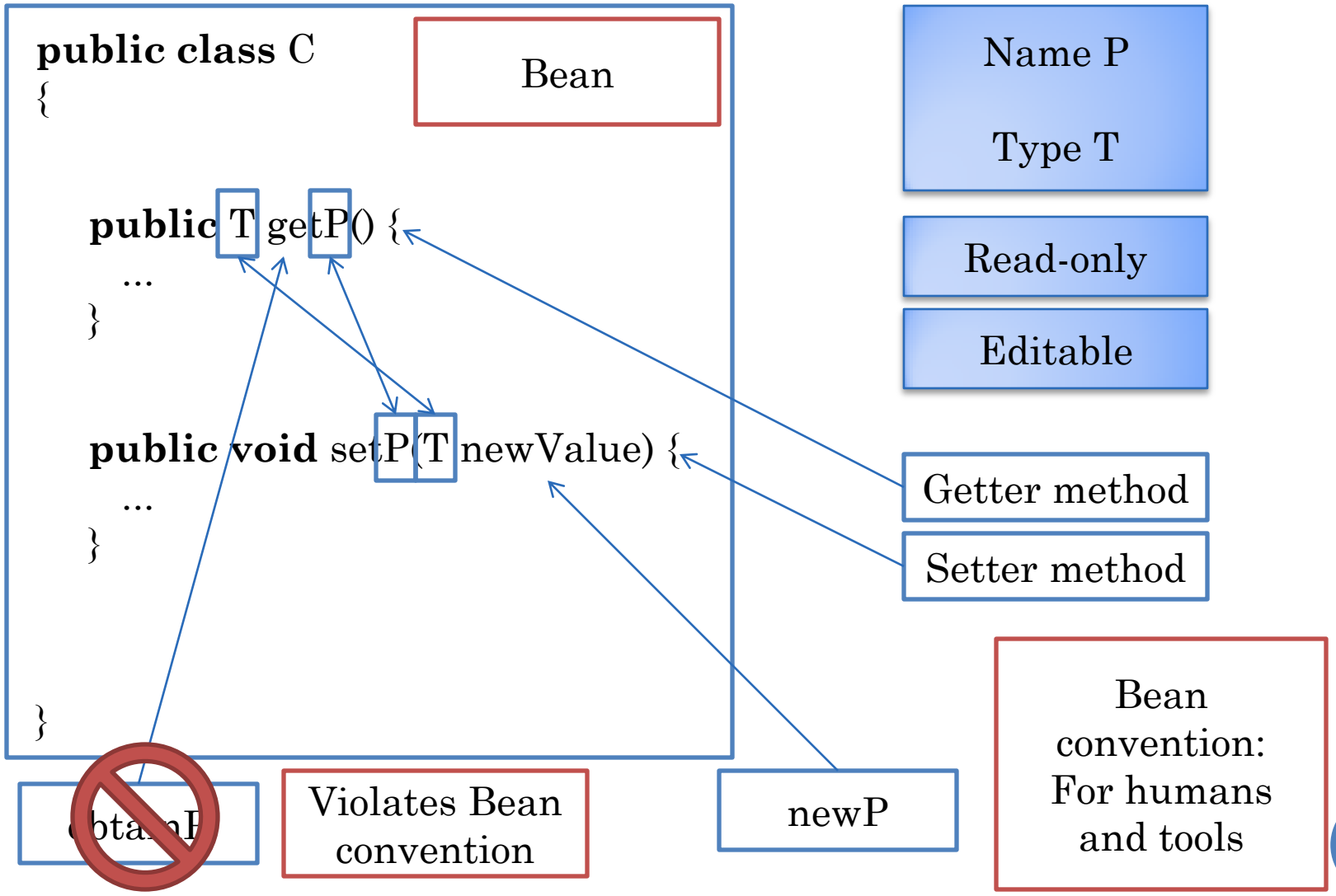


Properties

```
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);
        }
}
```

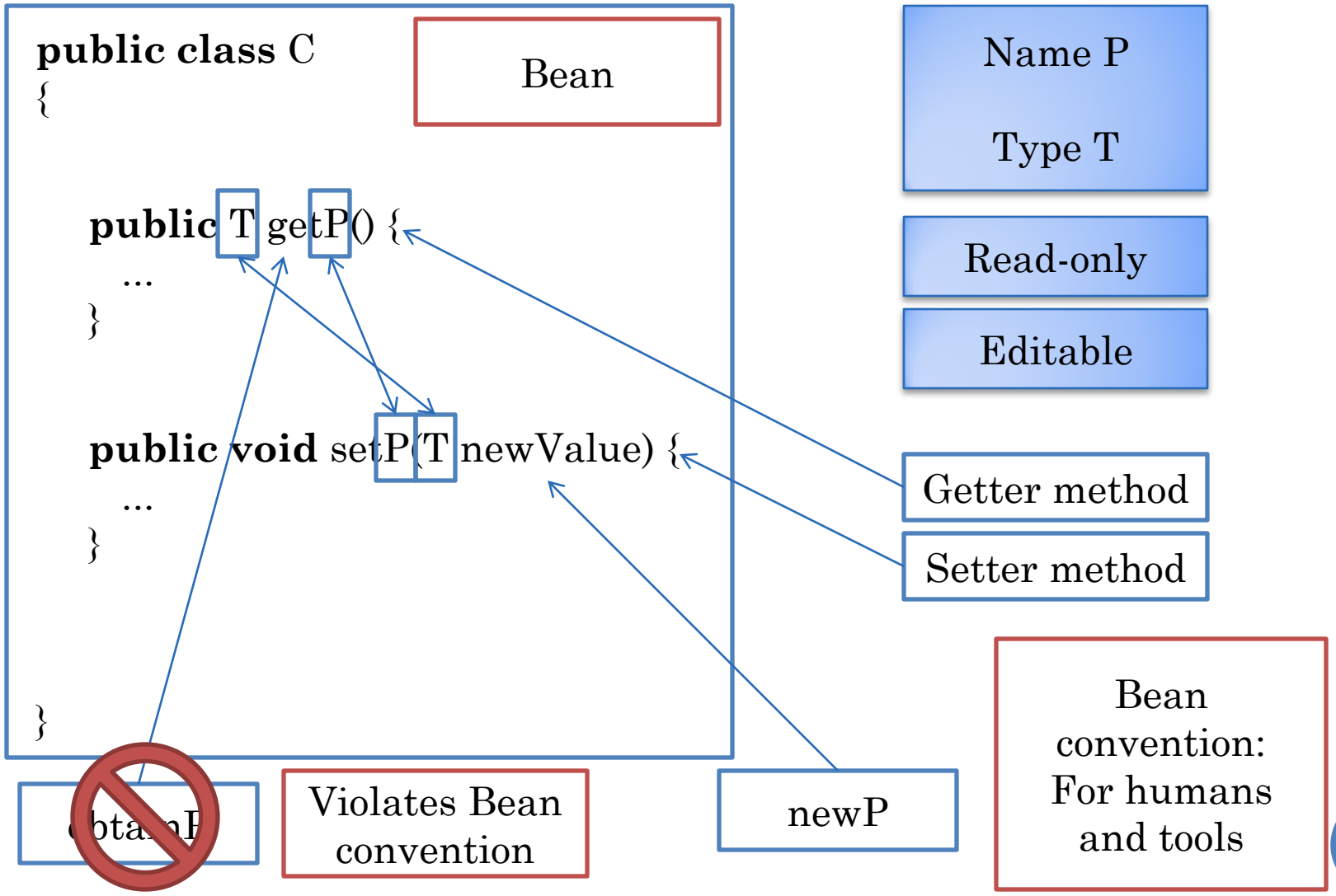
Read-only and Editable Properties

Typed, Named Unit of Exported Object State



Read-only and Editable Properties (review)

Typed, Named Unit of Exported Object State



Properties

Height

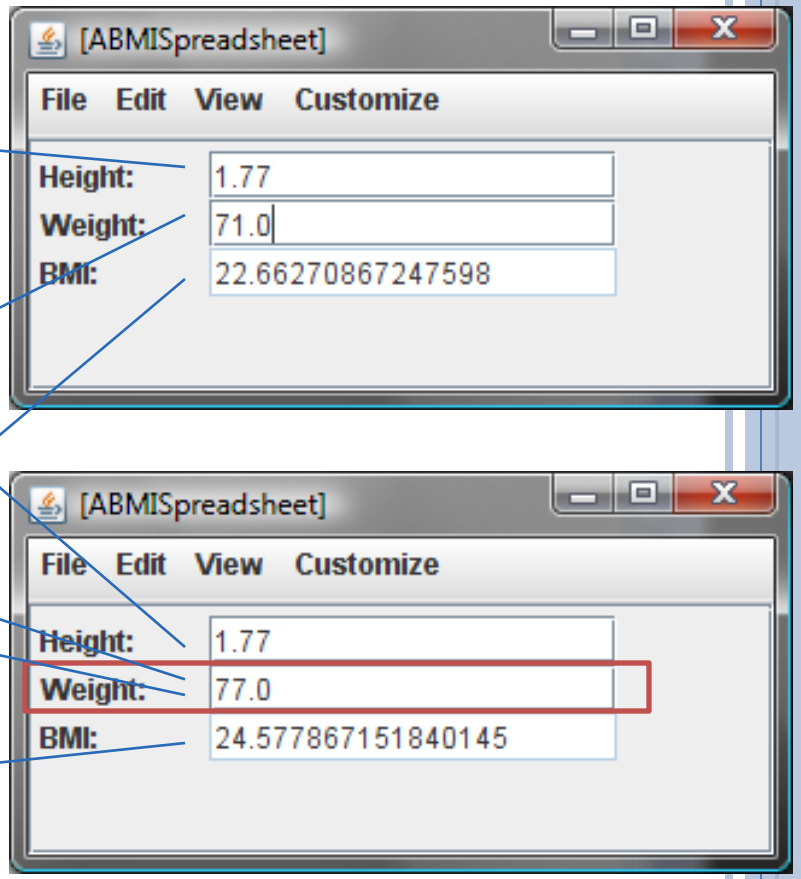
Weight

BMI

```
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);
        }
}
```

ObjectEditor Property Manipulation

```
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);
    }
}
```

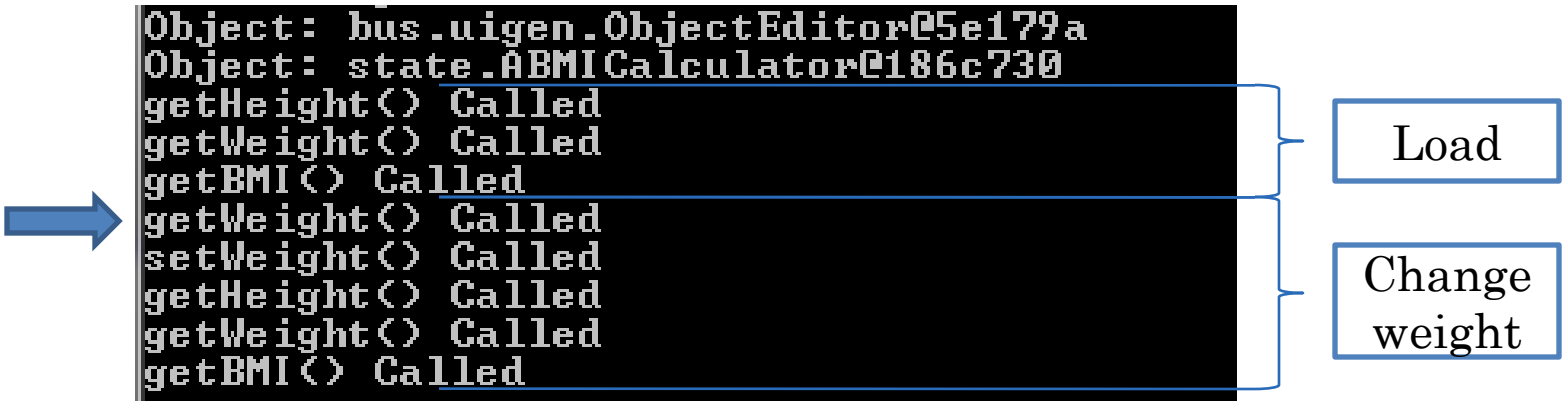


Tracing Method Calls

```
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);
    }
}
```

Debug output

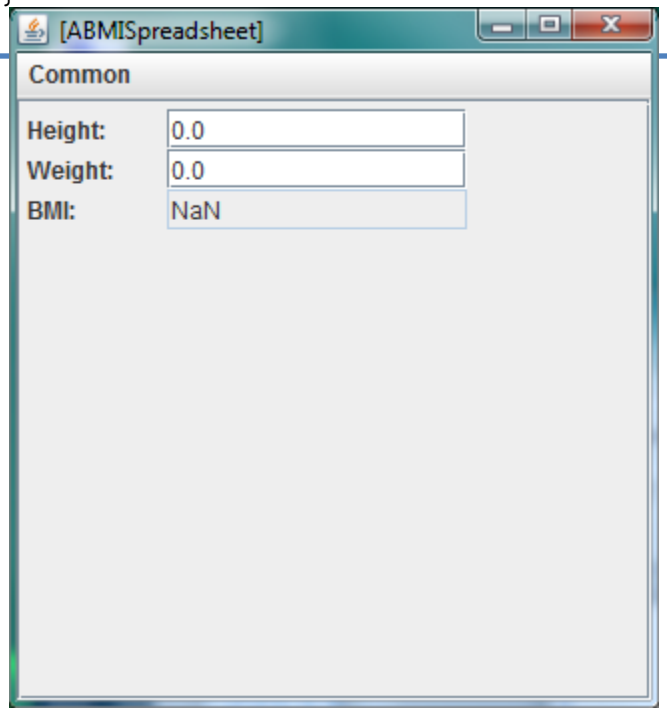
Actual Trace



Extra getWeight() call made by the undo-redo mechanism in ObjectEditor

Displaying and then Changing Object

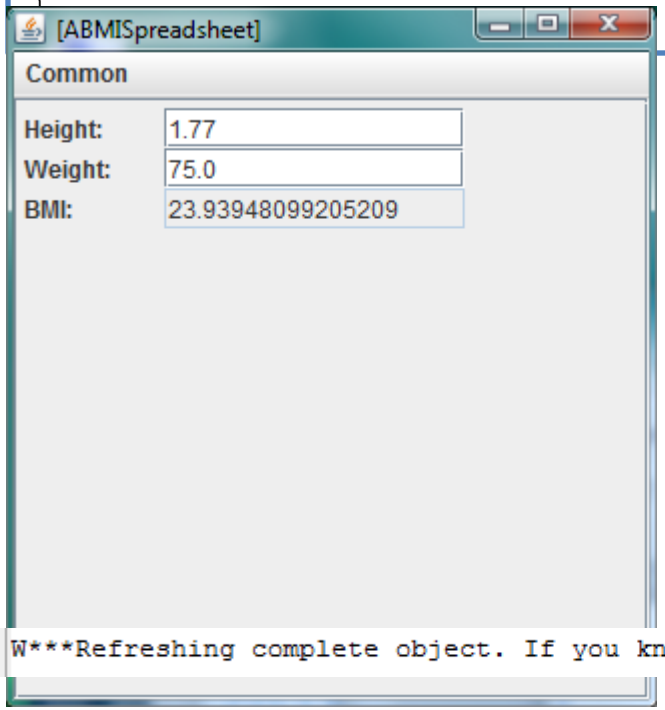
```
public class ABMISpreadsheetManipulatedByMainAndObjectEditor {
  public static void main (String[] args) {
    ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    ObjectEditor.edit(bmiSpreadsheet);
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
  }
}
```



Setters not called through ObjectEditor, so it does not know it should refresh

Refreshing ObjectEditor from Main

```
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();
  }
```

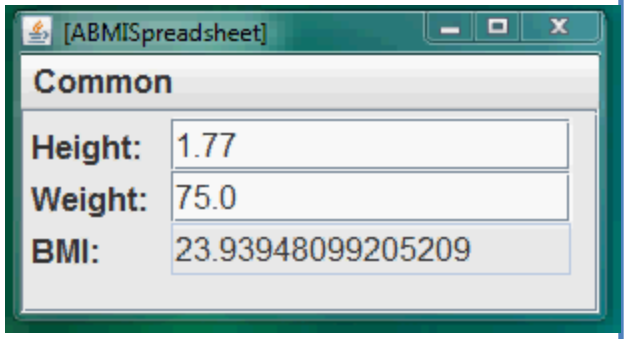


Better ways to refresh we will learn later

```
W***Refreshing complete object. If you know them, annonce property and/or list events.
```

Demoing Object

```
public class ABMISpreadsheetAnimatingDemoer {
  public static void main (String[] args) {
    ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
    OEFrame 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

Properties Classification

```
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);
        }
}
```

Height	Editable Independent Stored
Weight	Editable Independent Stored
BMI	Read-only Dependent Computed

Properties?

HowTall

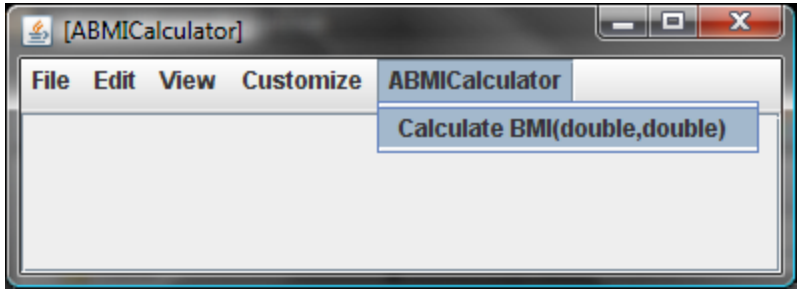
Weight

BMI

```
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);
        }
}
```

Properties?

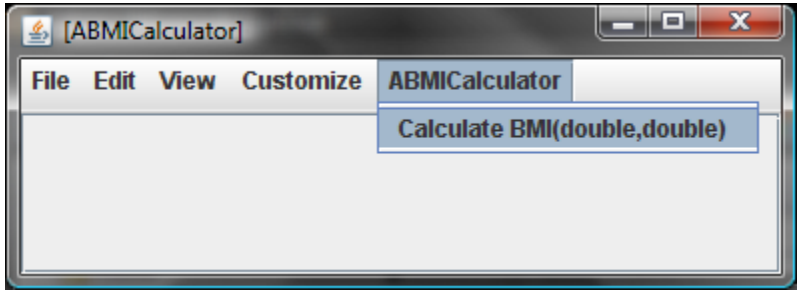
```
public class ABMICalculator  {
    public double calculateBMI (double weight,
                                double height) {
            return weight/ (height * height);
    }
}
```



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.

Bean Pattern Annotation

```
import util.annotations.StructurePattern
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheet {
        double height;
        public double getHeight() {
                return height;
        }
        public void setHeight(double newHei
                height = newHeight;
        }
        double weight;
        public double getWeight() {
                return weight;
        }
        public void setWeight(double newWe
                weight = newWeight;
        }
        public double getBMI() {
                return weight/(height*height)
        }
}
```

Why pattern annotation?

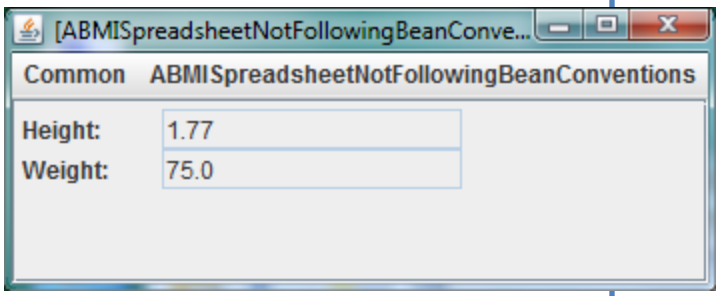
Documentation

Efficiency: OE does not need to look for pattern

Errors/warnings: Can give error message if pattern not followed

Error?

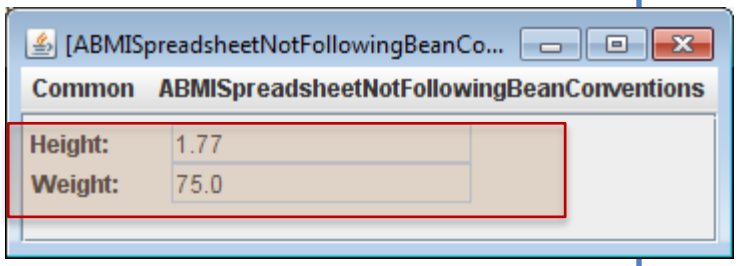
```
public class ABMISpreadsheetNotFollowingBeanConventions  {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
  public void set(double newWeight, double newHeight) {
    weight = newWeight;
    height = newHeight;
  }
  public double getHeight() {
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



Can system catch these errors?

(Editable) Property Name Annotations

```
import util.annotations.EditablePropertyNames;
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,
    weight = newWeight;
```

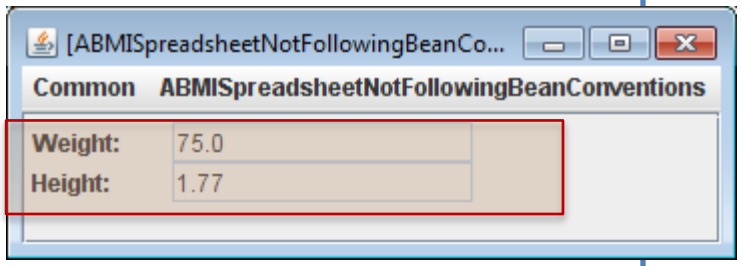



```
E***For property: height in editable property names, please define a setter with the header:
       public void setHeight(double <parameter name>)
E***For property: weight in editable property names, please define a setter with the header:
       public void setWeight(double <parameter name>)
E***For property: BMI in  property names, please define a getter with the header:
       public <T> getBMI()
```

```
    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,
    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

String

double

Two different operations with the same name

public void println(String val) {…}

public void println(**double** val) {…}

System.out.println("setWeight called");

System.out.println(newWeight);

More on Println

```
Object: bus.uigen.ObjectEditor@5e179a
Object: state.ABMICalculator@1deeb40
getHeight() Called
getWeight() Called
getBMI() Called
getWeight() Called
setWeight() Called: 165.0
getHeight() Called
getWeight() Called
getBMI() Called
```

System.out.println("setWeight called");

System.out.println(newWeight);

Operator Overloading

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

5 + 6

Ambiguous Context

Time flies like an arrow.

Fruit flies like an orange.

Operation Definitions

Java cannot use context to disambiguate

public void println(String val) {…}

public void println(String val) {…}

System.out.println("setWeight called");

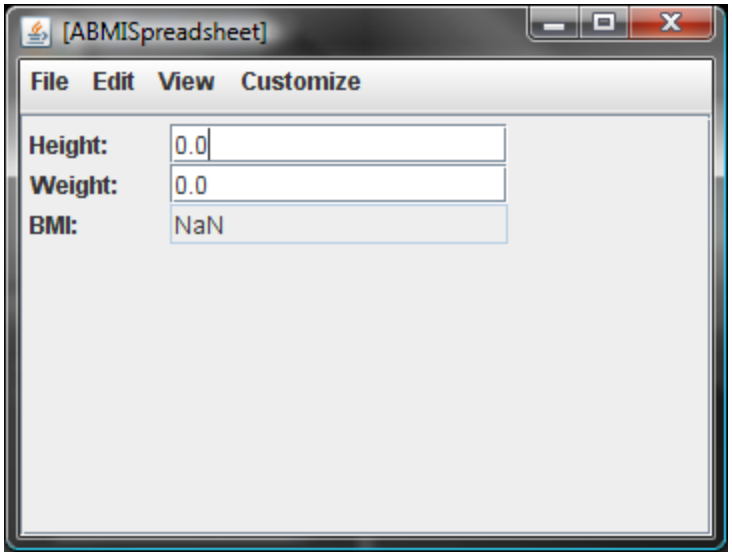
System.out.println(newWeight);

Defining two versions of a method ?

Why is overloading useful?

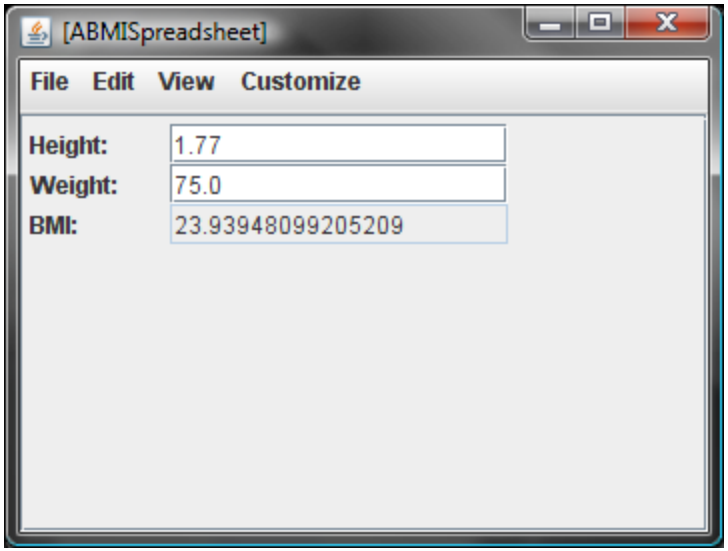
Inconsistent BMI State

```
ObjectEditor.edit(new ABMISpreadsheet());
```



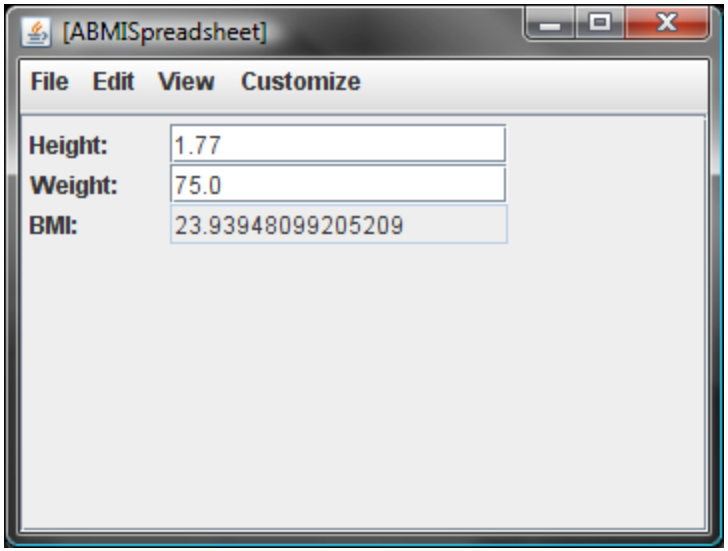
Fixing Inconsistent BMI State

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



Always Consistent BMI State

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



Constructor

```
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);
        }
}
```

Calling setter methods instead of modifying variable directly makes debugging easier as you can set breakpoint on setter to trap writes to it

Constructor

```
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);
        }
}
```

Constructor name must be the name of the class

Constructor name is also the type of object returned

Every Class has a Constructor

```
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);
        }
}
```

Equivalent Class Created by Java

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

        public ABMISpreadsheet() { }

        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);
        }
}
```

Inserted in Object Code not in Source Code

If Programmer Specifies no Constructor, Java inserts a null constructor

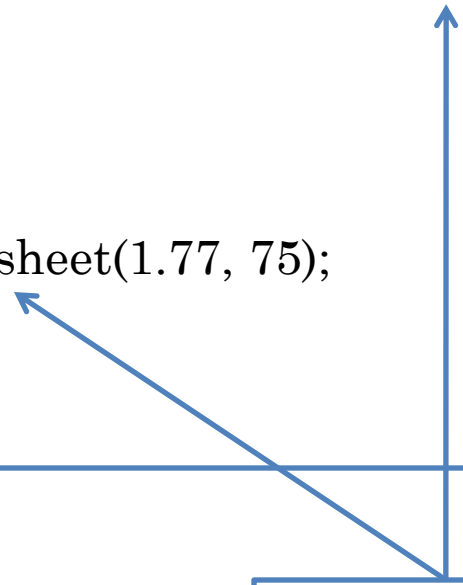
A Class Can Have Multiple Constructors

```
public class ABMISpreadsheet {
        double height, 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);
        }
}
```

Overloaded Constructor

Using Overloaded Constructors

```
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

```
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);
```

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 sb

Reassigning a new object less efficient

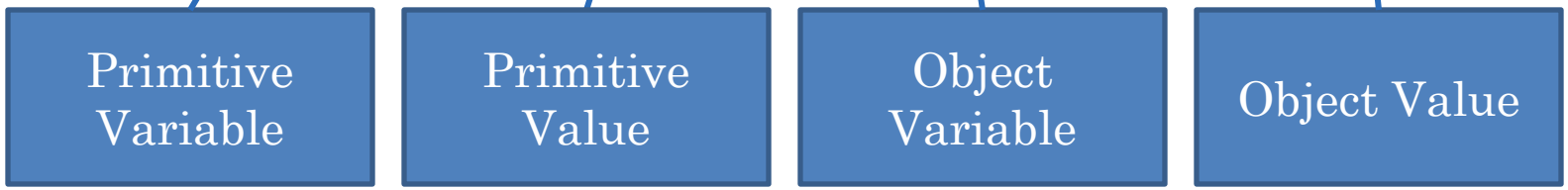
Changes the object to which sb points

Programmatic Property Manipulation

```
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 );
    }
}
```

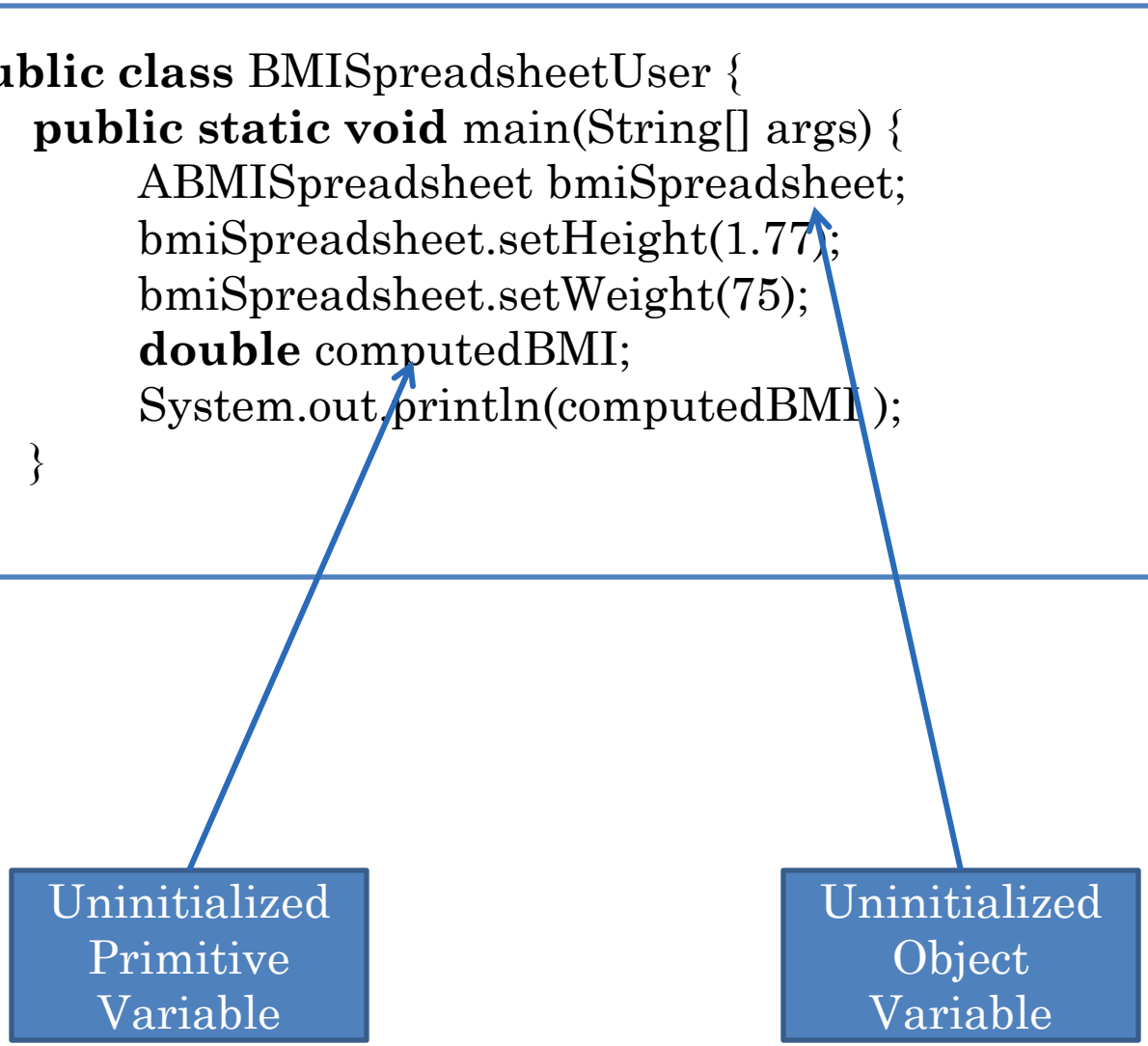
Objects vs. Primitives

```
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 );
    }
}
```



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 );
    }
}
```



Default Values for Variables

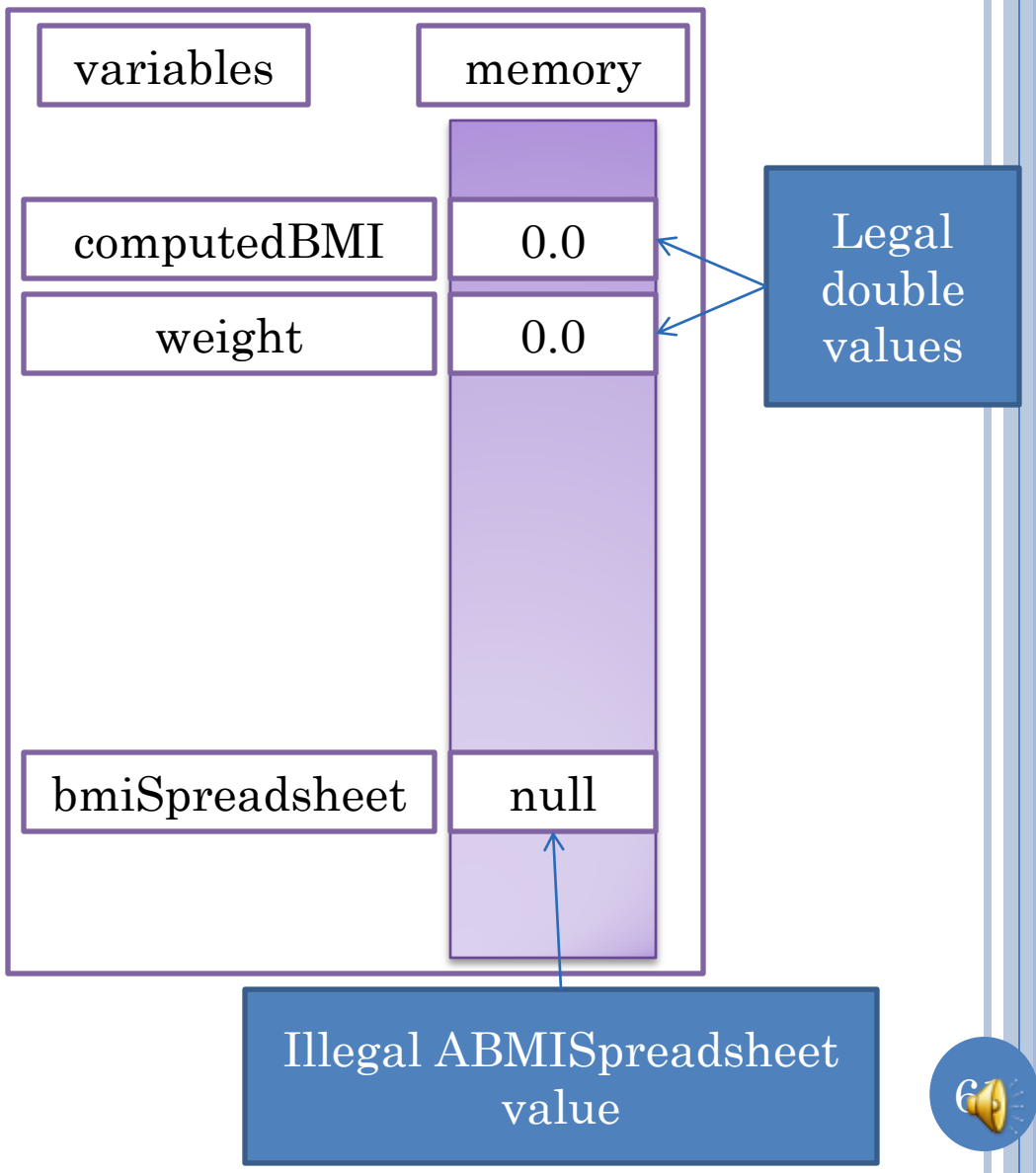
Primitive Variables

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

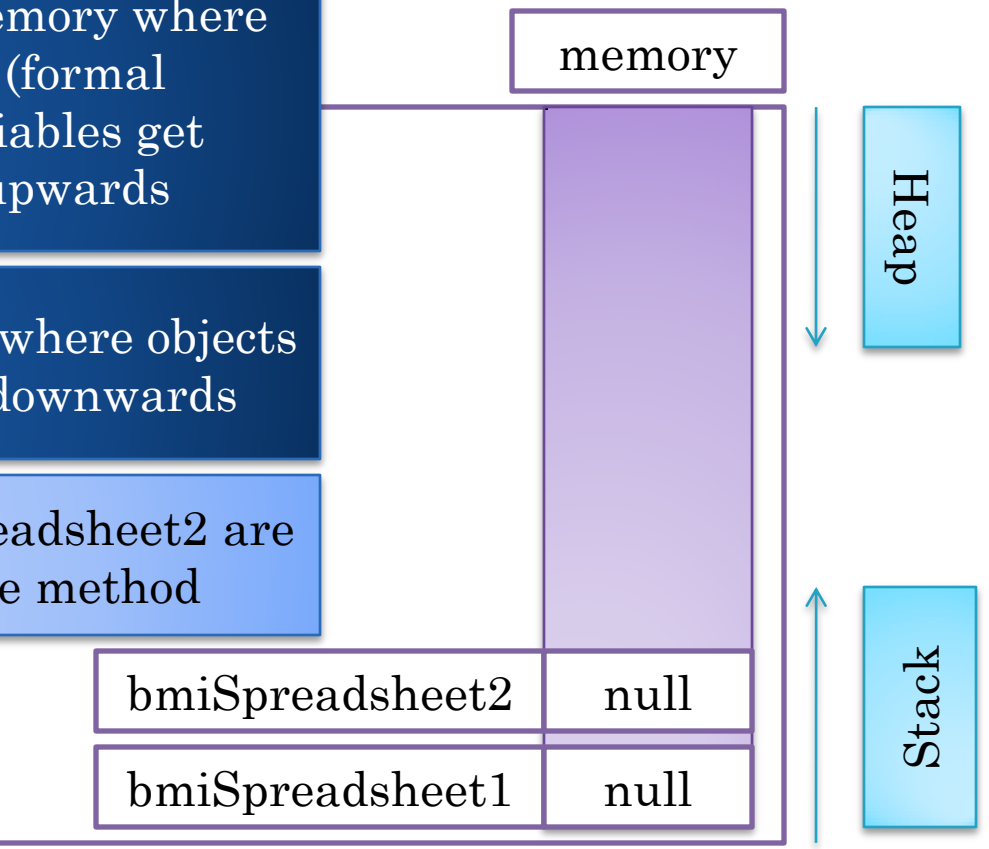
Memory Representation

```
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

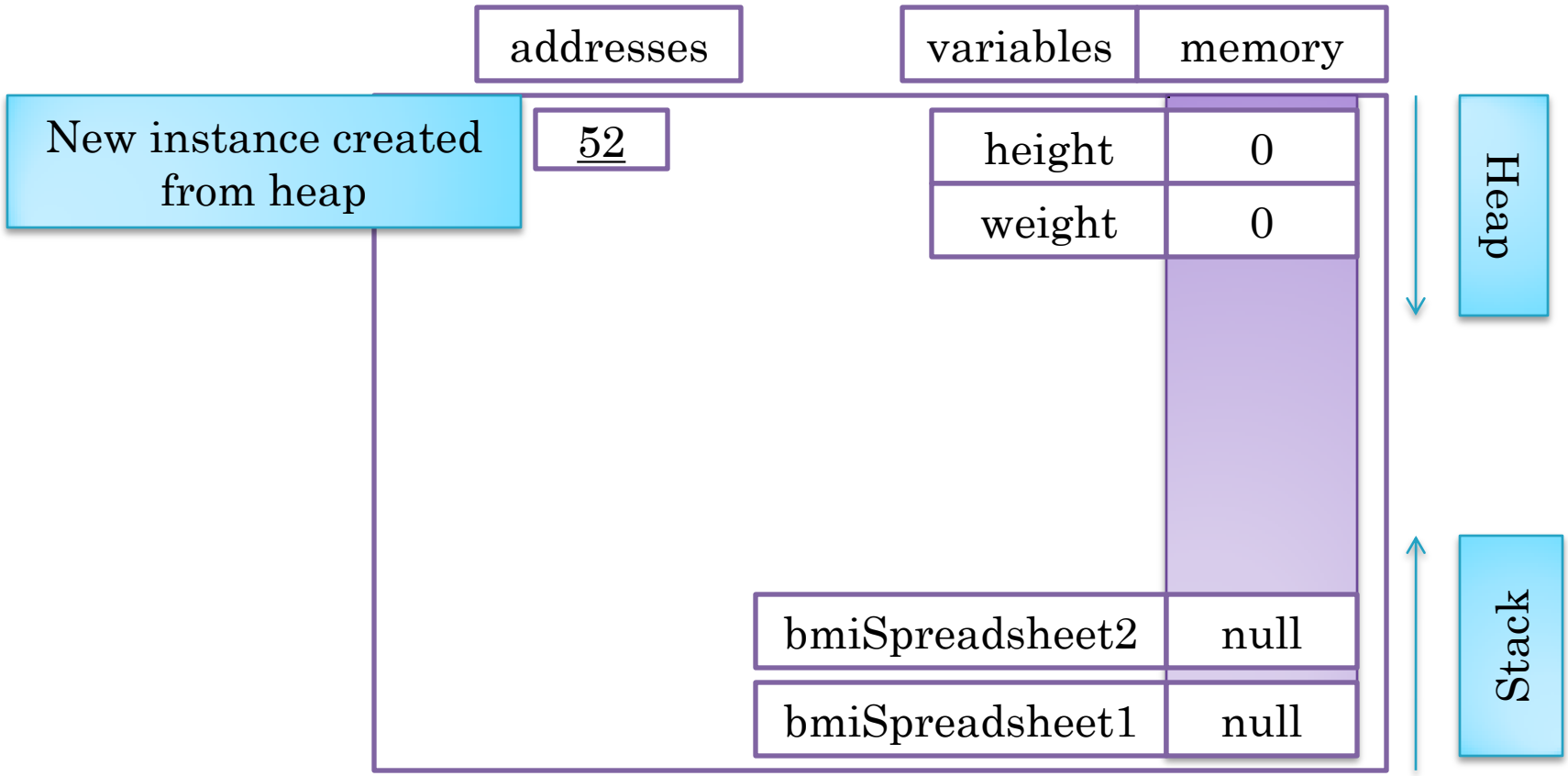
Heap: area at top of memory where objects get allocated, heap grows downwards

bmiSpreasheet1 and bmiSpreadsheet2 are internal variables of some method



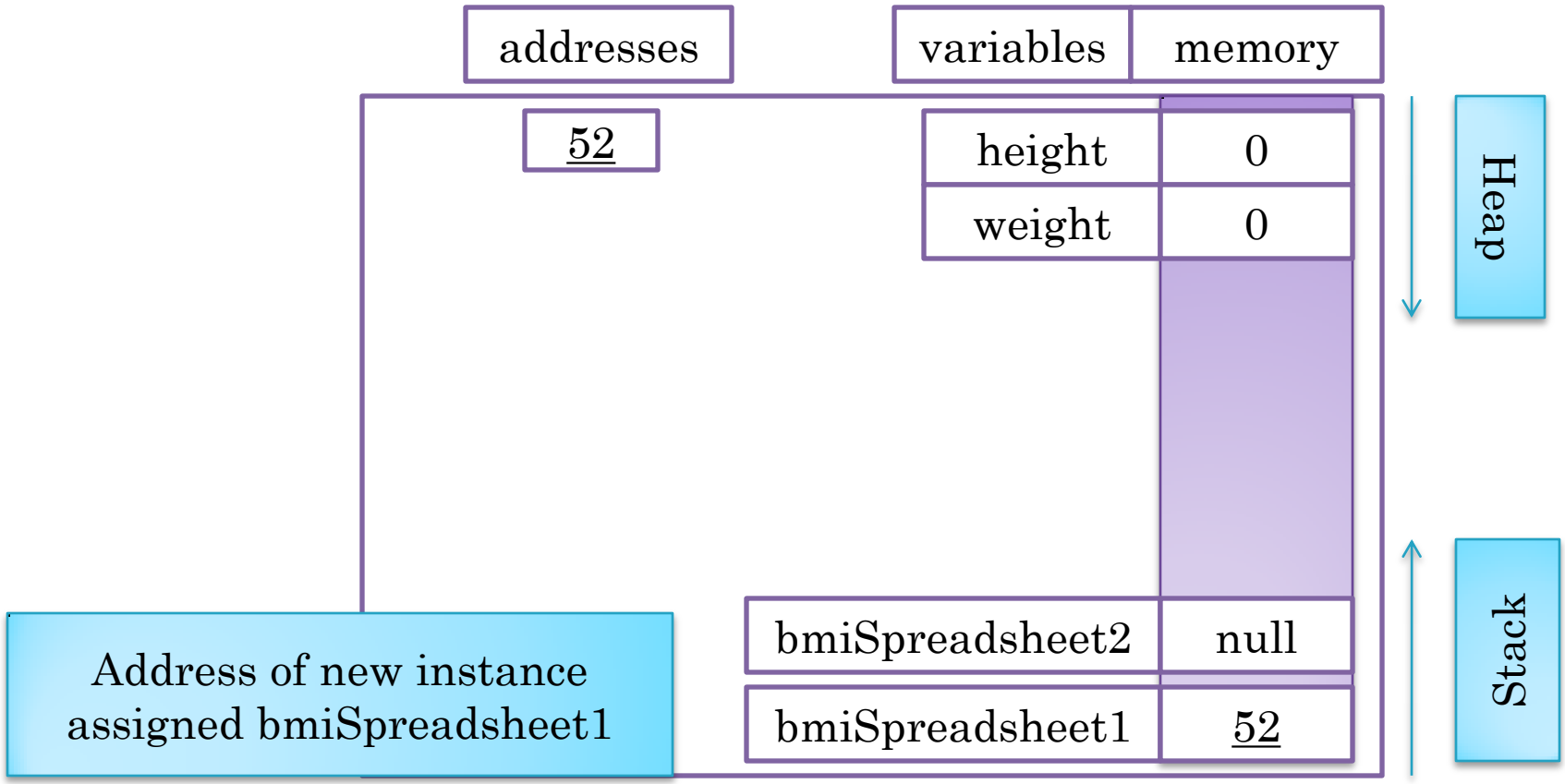
New Instance created

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



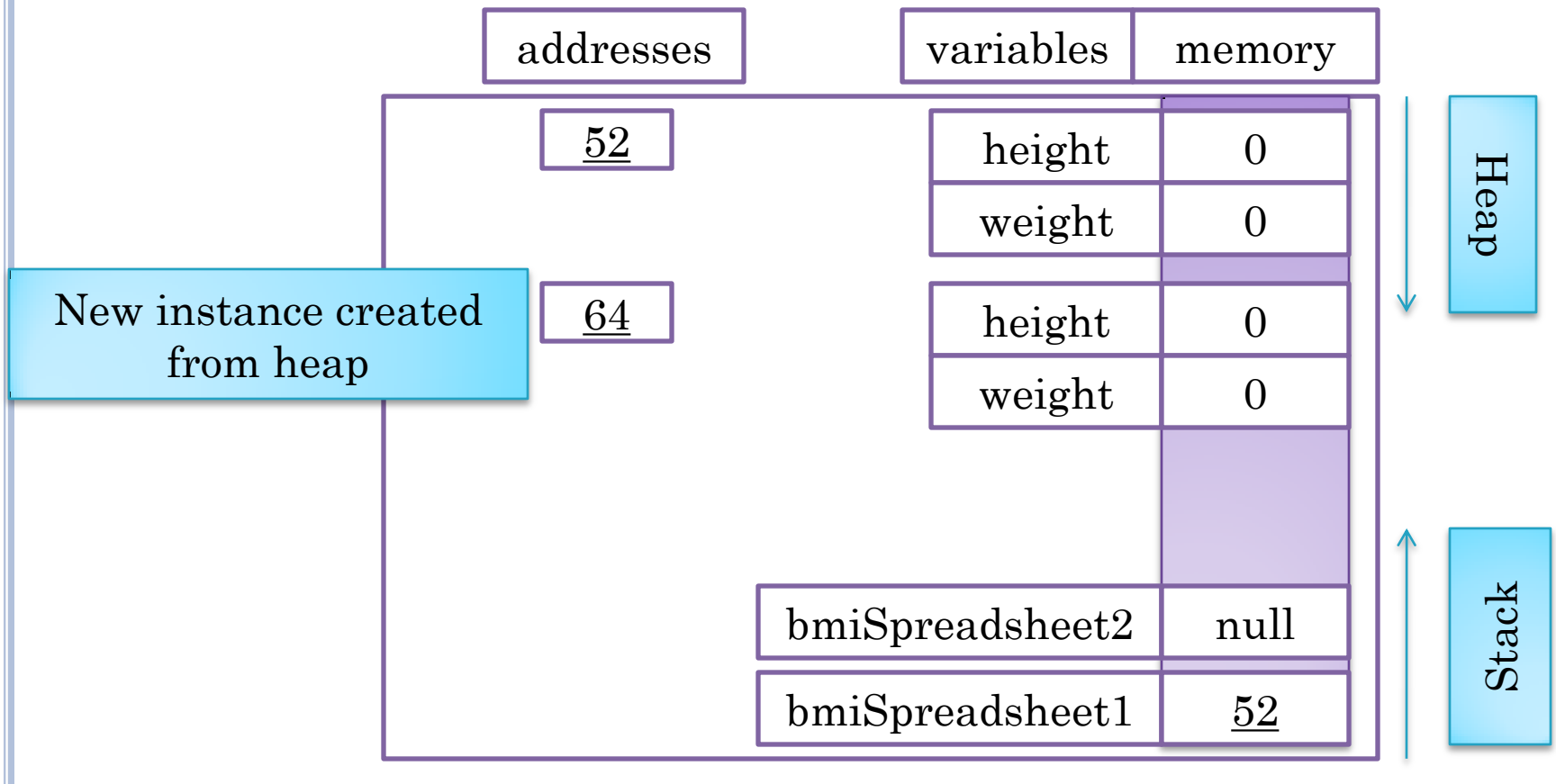
Object Assignment

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



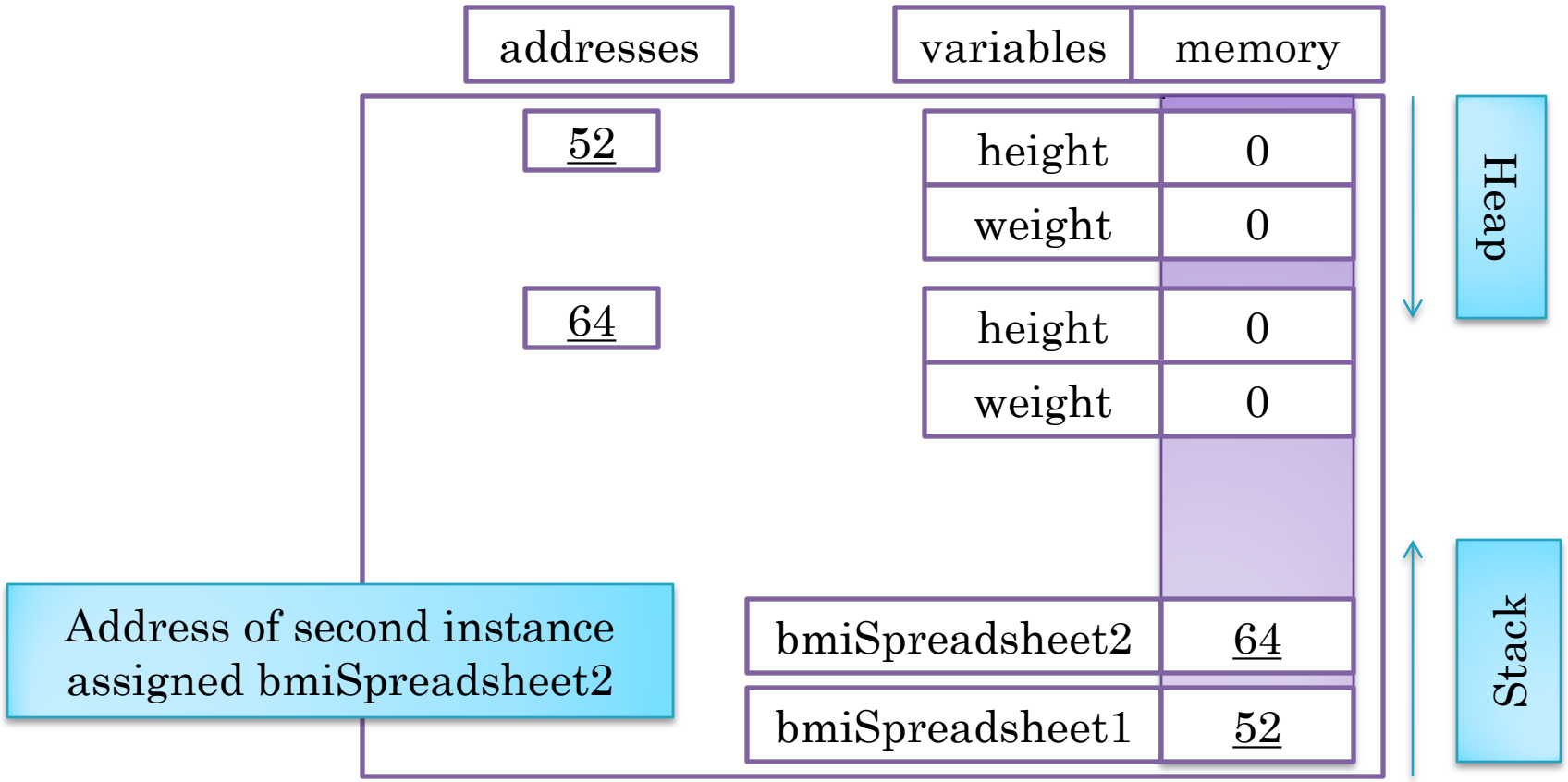
Second Instance Created

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



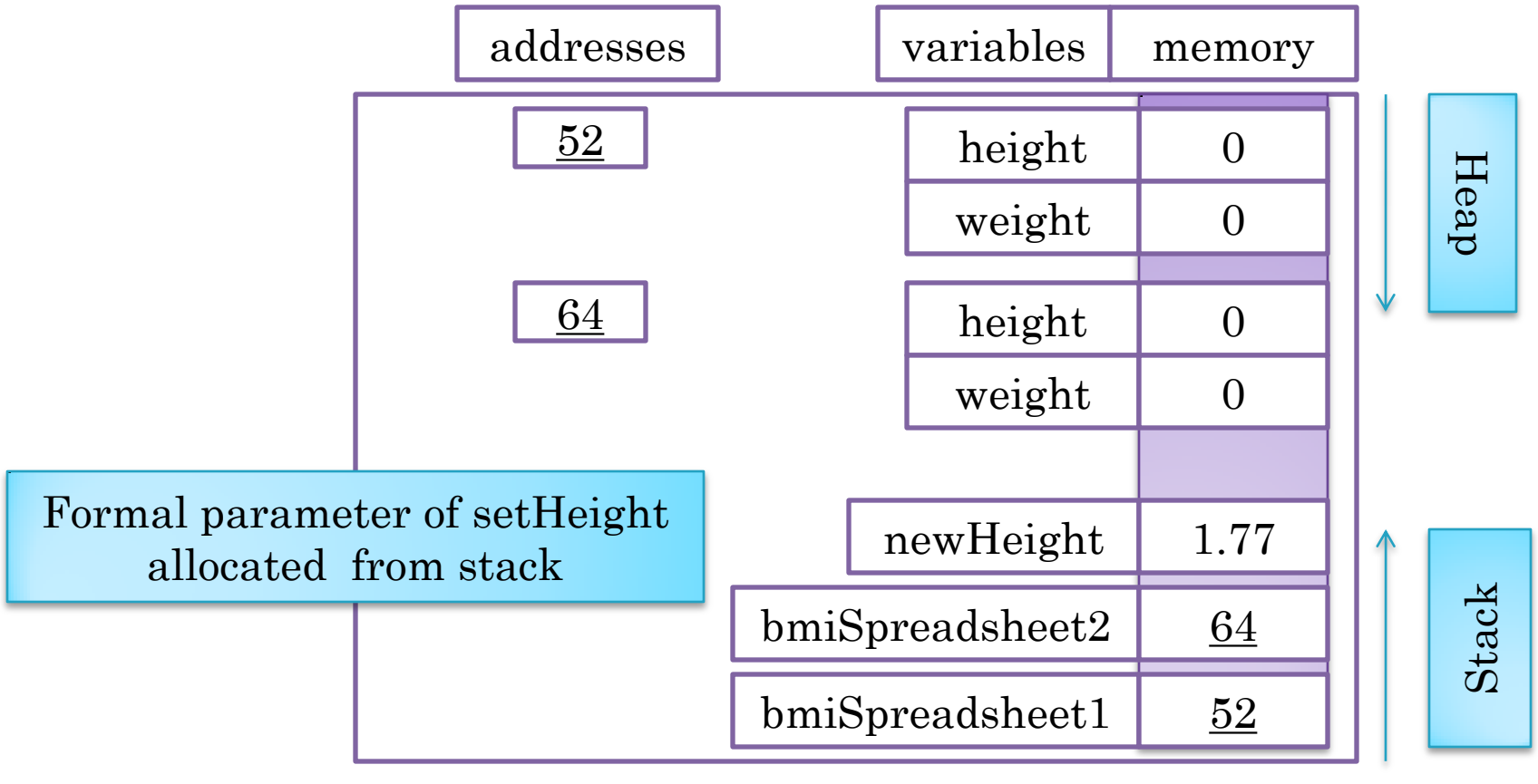
Second Assignment

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



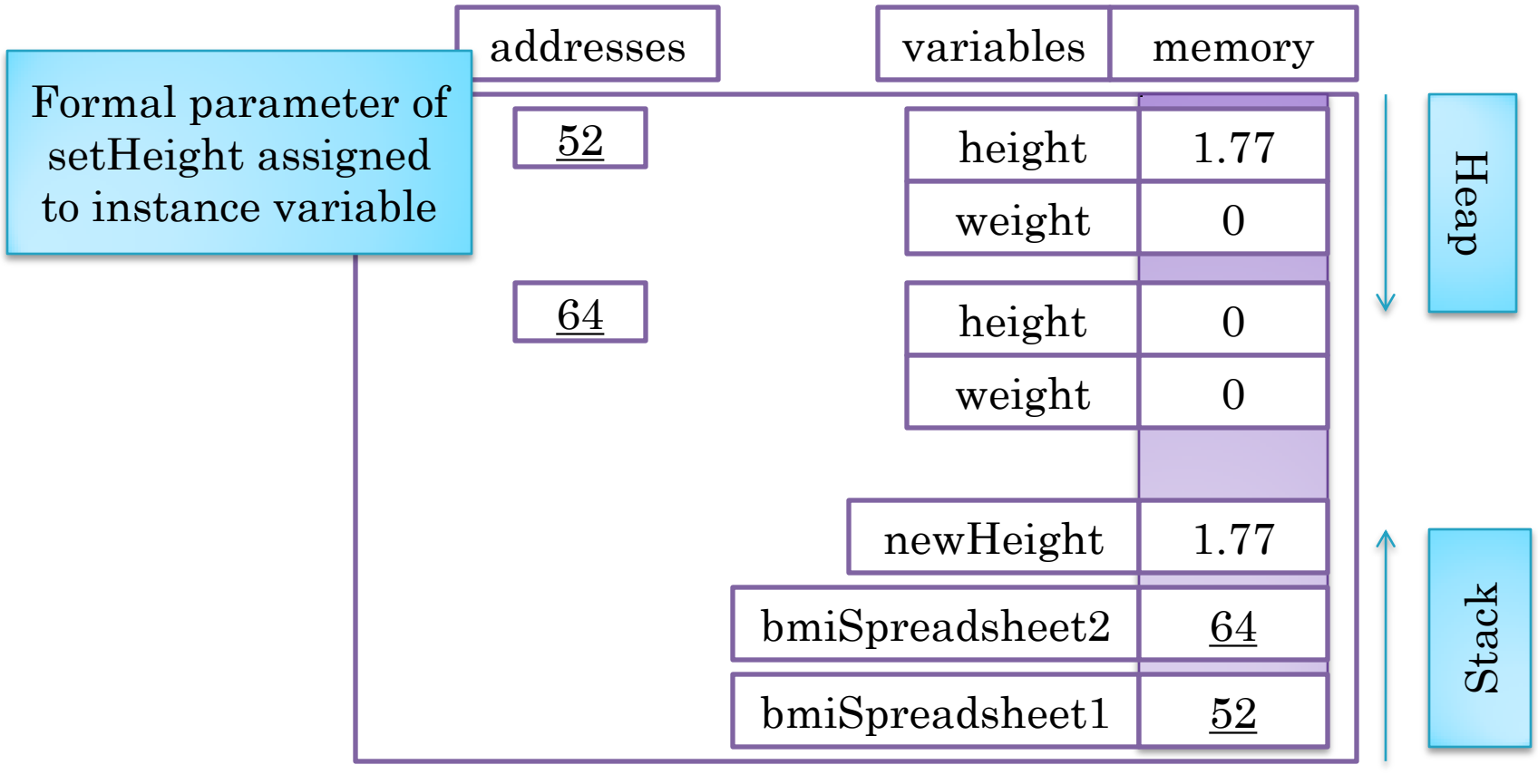
START OF SETHEIGHTCALL

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



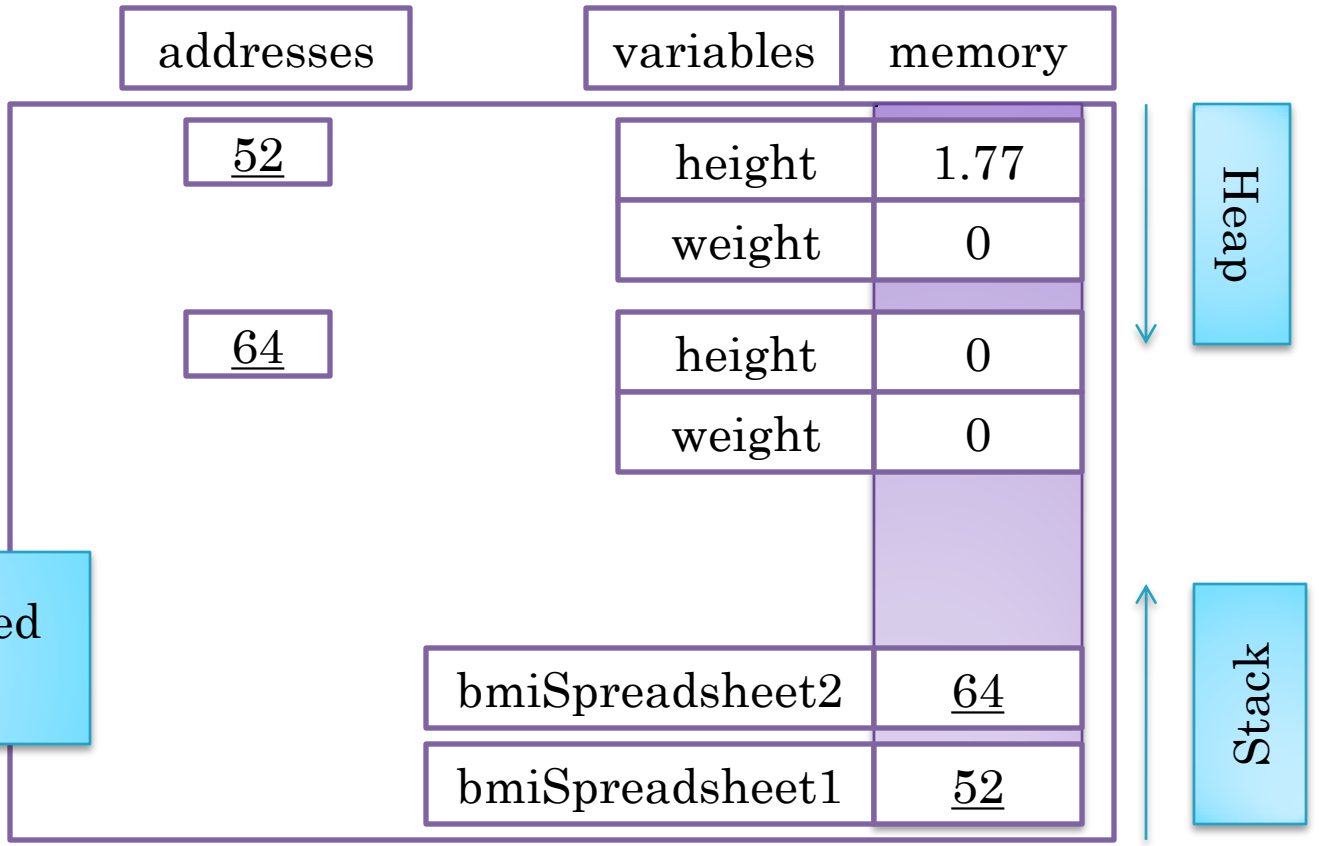
After setHeight Assignment

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



setHeight Returns

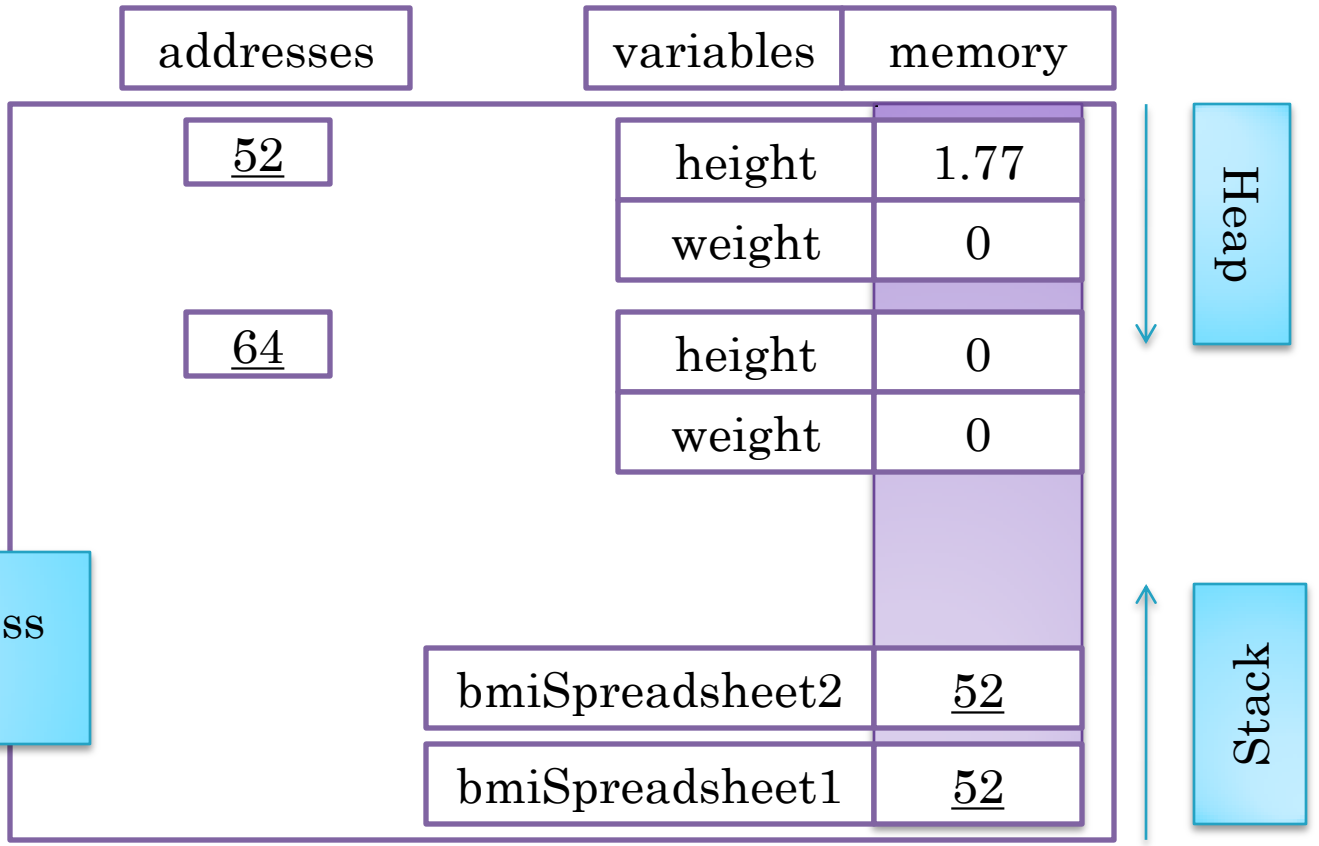
```
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77)
bmiSpreadsheet2 = bmiSpreadsheet1;
```



newHeight popped from stack

After Assignment

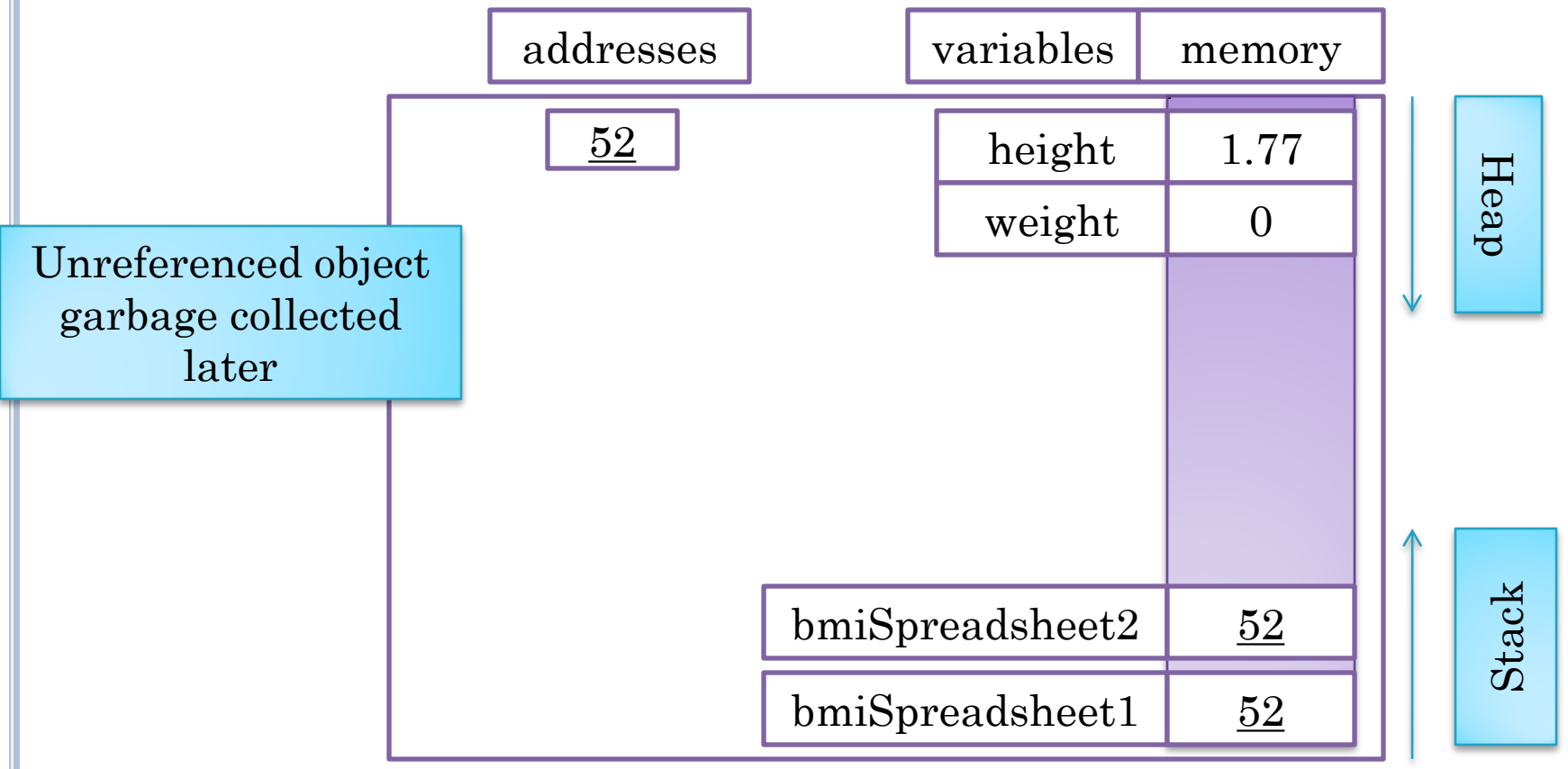
```
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77)
bmiSpreadsheet2 = bmiSpreadsheet1;
```



== does an address (pointer) copy

After Garbage Collection

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

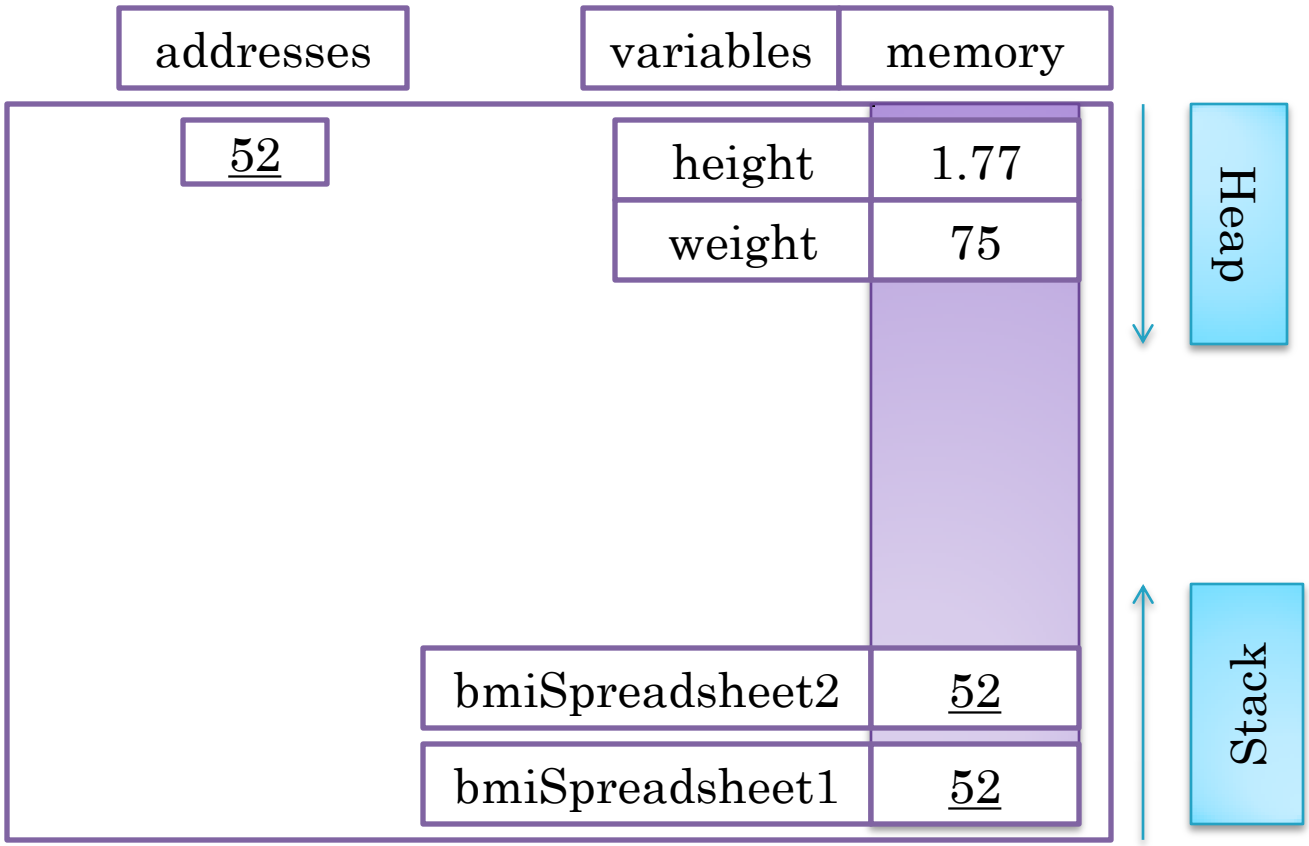


Assignment is not a copy

```
bmiSpreadsheet2 = bmiSpreadsheet1;
bmiSpreadsheet1.setWeight(75);
```

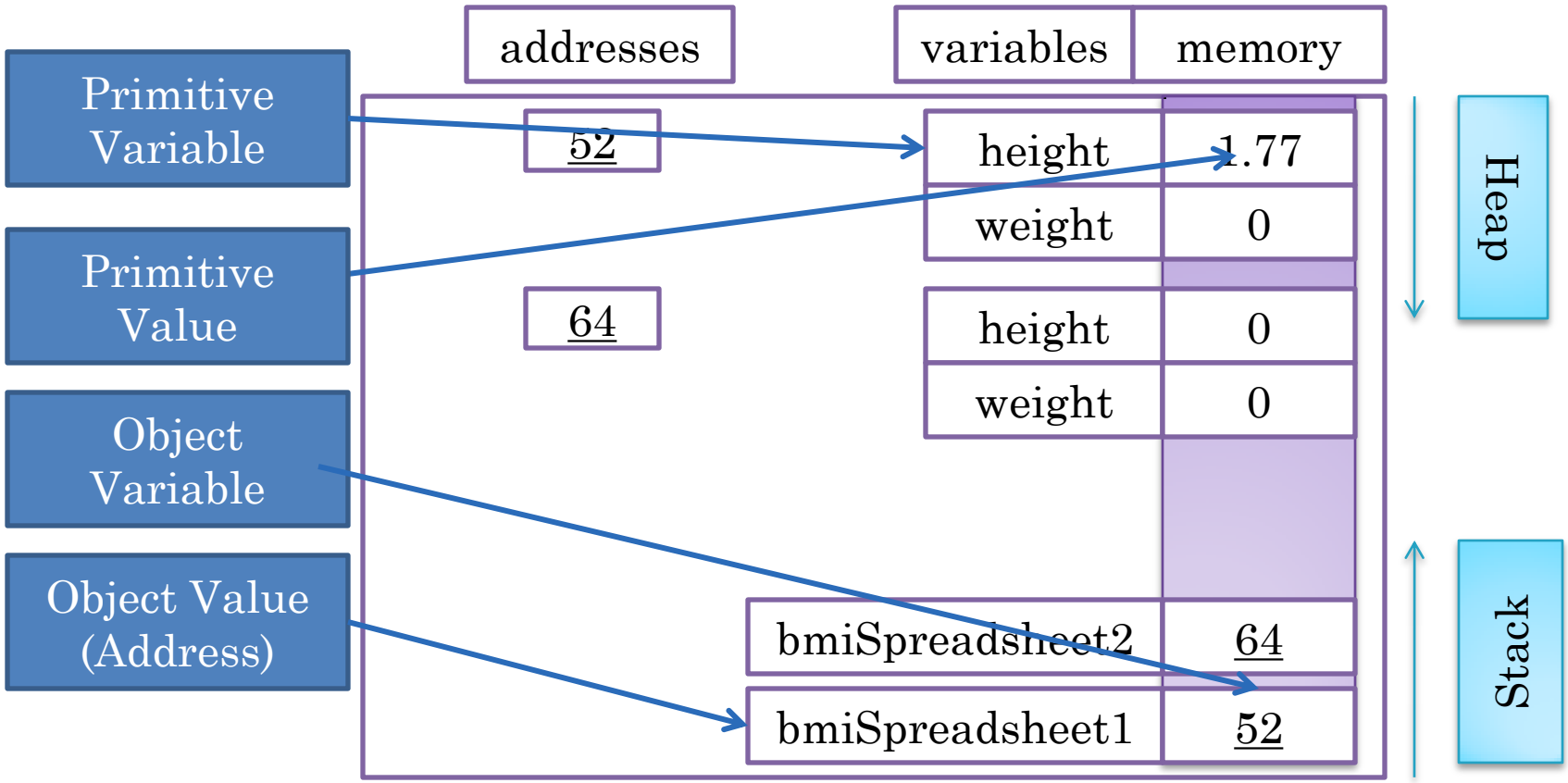
```
bmiSpreadsheet1.getWeight() ==
bmiSpreadsheet1.getWeight()
```

→ true



Primitives vs. Objects Again

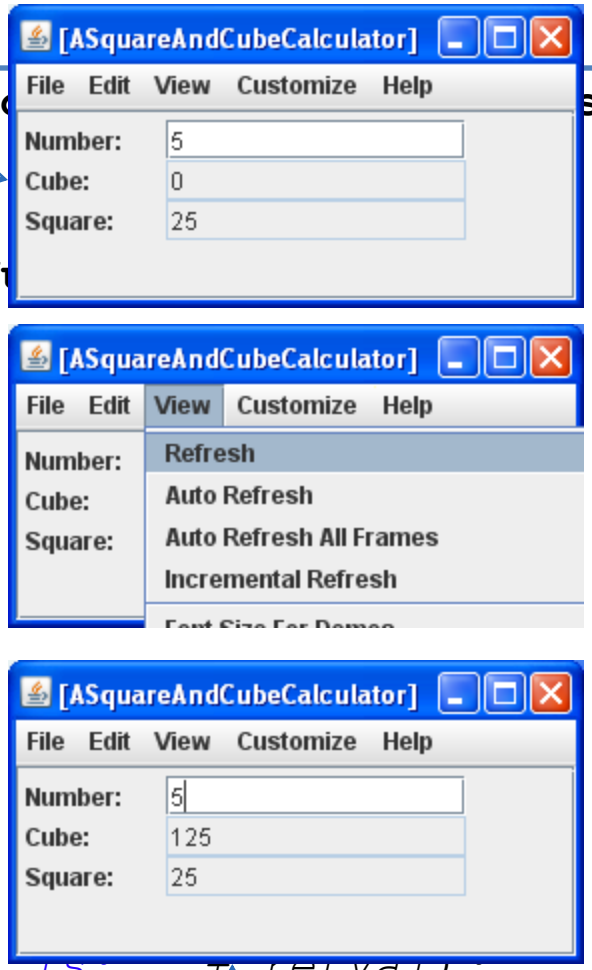
```
ABMISpreadsheet bmiSpreadsheet1 = new ABMISpreadsheet();
ABMISpreadsheet bmiSpreadsheet2 = new ABMISpreadsheet();
bmiSpreadsheet1.setHeight(1.77);
bmiSpreadsheet2 = bmiSpreadsheet1;
```



Functions with Side Effects

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

```
public void setNumber(int theNu
   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;
}
}
```



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

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

Make is easier to implement correct programs with threads

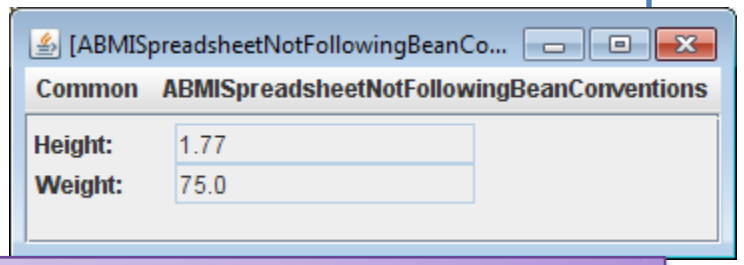
Make is easier to implement correct programs with threads and hashtables

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

Bean Pattern?

```
@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;
```



Warning if (editable) properties not declared?

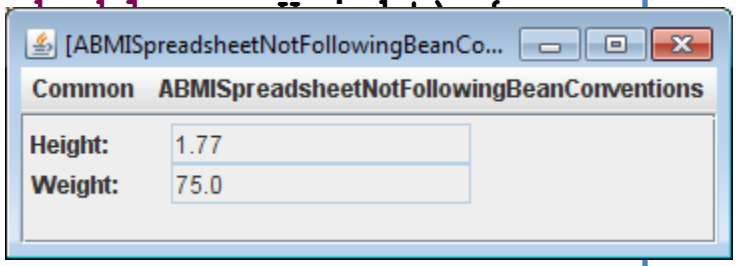
Overhead, chances of mistake low, C# has built in support for properties

Why warning if no structure annotation?

(Pattern)Annotation

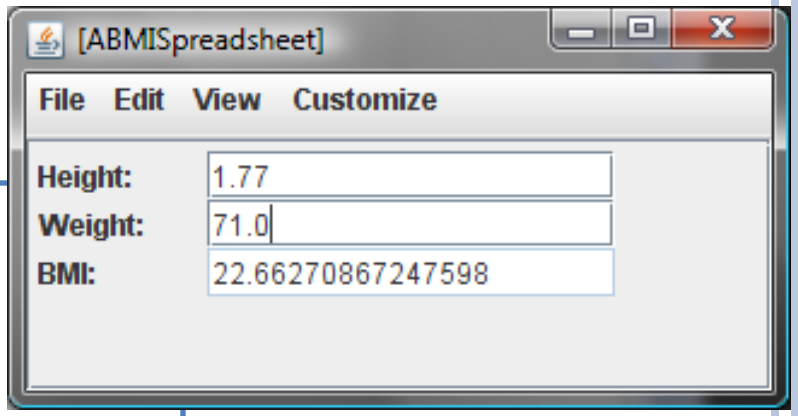
```
import util.annotations.StructurePattern;
import util.annotations.StructurePatternNames;
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheetNotFollowingBeanConventions   {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
```

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.

ObjectEditor Property Manipulation



Height

Weight

BMI

```
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);
        }
}
```