

# COMP1409: Introduction to Software Development I

Mike Mulder ([mmulder10@bcit.ca](mailto:mmulder10@bcit.ca))

Week 3

# Agenda

- Quiz
  - Quiz 2
  - Review Answers
- Lab 2 Feedback
- Lab 3B Questions
- OO Principles, Java Classes and Review
- Lesson 3 Topics
  - Methods
  - Console Output
  - String Concatenation
  - Main
- Lab 3
  - Create a “Complete” Java Class (Instance Variables, Constructor and Methods)

# Quiz 2

- Closed book, laptop, phone, etc.
- You have a maximum of 15 minutes to complete
- Raise your hand when you are done, and I will get your paper
- We will review the answers afterwards

# Lab 2 – General Feedback

- Remember to format your code (Edit -> Auto-layout in BlueJ)
- Assignment and Equality are different:
  - `a = b` // Assigns the value of b to a
  - `(a == b)` // true if a is equal to b and false otherwise
- A boolean (i.e., `isFiction`) can only be true or false. There is no need to check if the parameter is true or false before setting the instance variable.
- There several ways to do the weight check:
  - `if ((theWeight >= 0.0) && (theWeight <= 10.0))`
  - `if (!(theWeight < 0.0) || (theWeight > 10.0))`

# Lab 3B Questions

- Available Monday and Tuesday evenings for questions by e-mail
- E-mail: [mmulder10@bcit.ca](mailto:mmulder10@bcit.ca)

# Object Oriented Principles

Abstraction

Encapsulation (or Information Hiding)

Inheritance

Polymorphism

We are only covering ***Abstraction*** and  
***Encapsulation*** in COMP 1409

# Abstraction

Java objects represent real objects, but they are not real objects. They are **models** of real objects.

The model is an **abstraction** of a real concept.

**Abstraction** means dealing with the level of detail that is most appropriate to a given task. It is the process of extracting a public interface from the inner details.

## Car

Use for inventory of cars on a used car lot.

## Car

Used for the state and location of a car in a production line

## Car

Used to store the current sensor values for a car in an engine management system

# Encapsulation

Also known as **Information Hiding** in Object Oriented Programming.

Each class has a **public interface**: The collection of attributes and methods that other objects can use to interact with that object.

**Encapsulation**: This process of hiding the implementation, or functional details, of an object.

How is encapsulation implemented in the Java Programming Language?



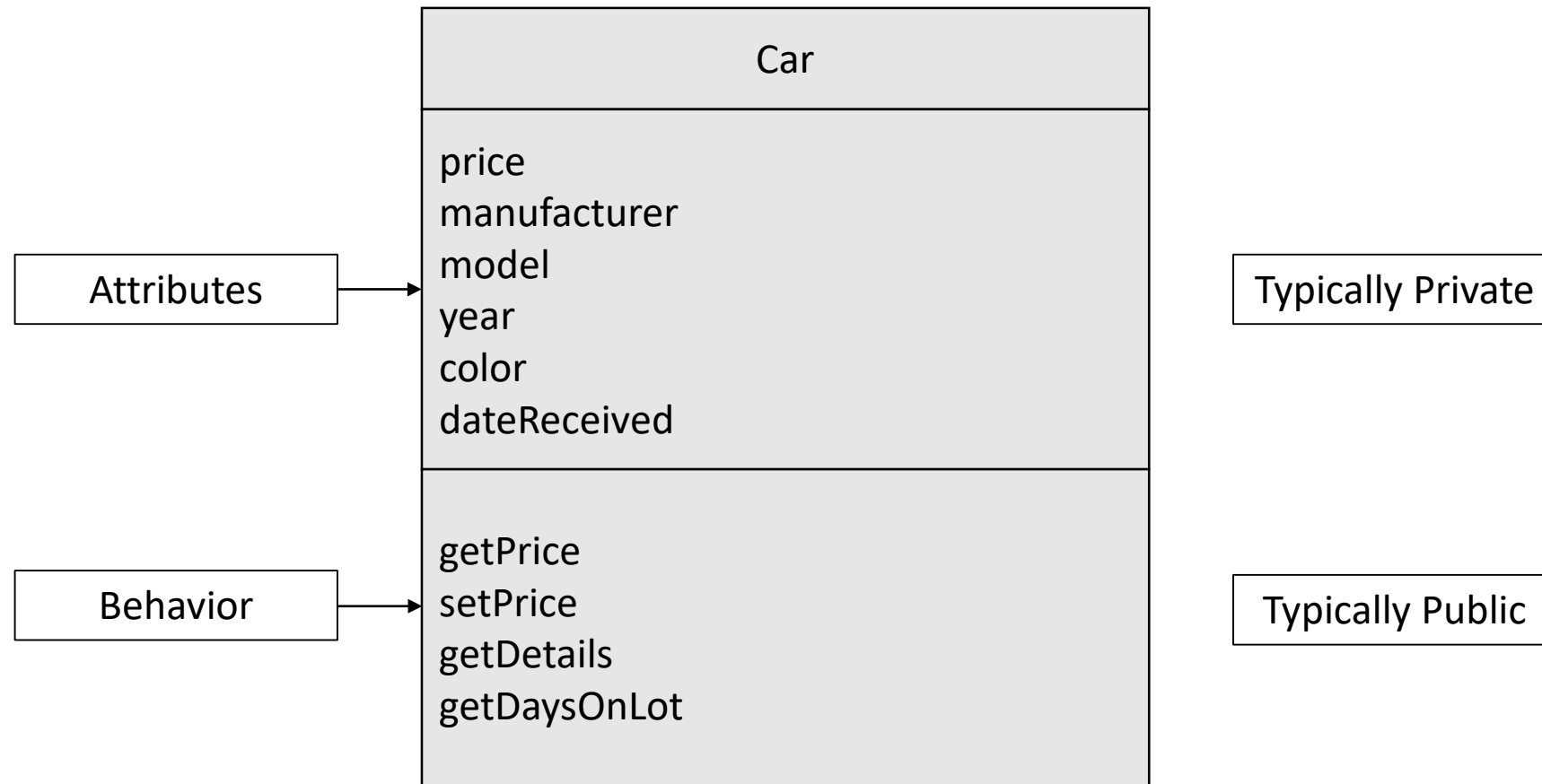
# Public Interface

The Public Interface needs to be carefully designed as it is difficult to change it in the future.

Changing the interface will break any client objects that are calling it.

We can change the internals all we like, for example, to make it more efficient, or to access data over the network as well as locally, and the client objects will still be able to talk to it, unmodified, using the public interface.

# Public Interface - Example




# Anatomy of a Java Class

```
/**
 * Two dimensional point.
 * @author Mike Mulder
 * @version 1.0
 */
public class Point {
    private int x = 0;
    private int y = 0;

    /**
     * Point constructor
     * @x x-axis coordinate
     * @y y-axis coordinate
     */
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    ???
}
```



**A – Class Name (and Visibility)**

**B – Instance Variables (including visibility and type)**

**C – Constructor (including visibility and parameters)**

**D – Methods (Today's Class)**

# Some More Review Questions

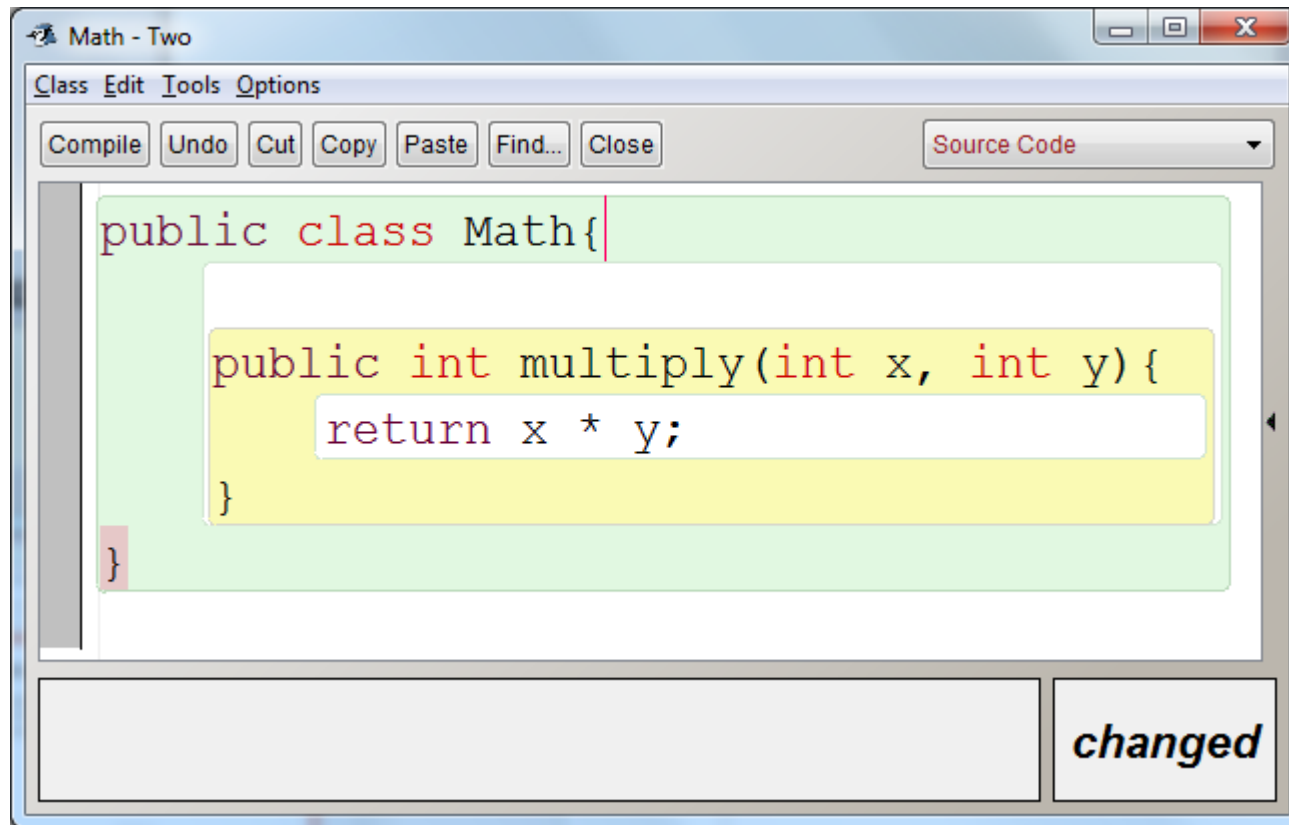
- What is the **this** keyword in a Java class used for?
- What does **null** mean with respect to object/reference data types? Does it apply to primitive types?
- What are the two types and three styles of comments?
- What does **throw new IllegalArgumentException("Bad Data")** do?

# Learning Outcomes: Lesson 3

- Methods: return types and parameters
- Accessors
- Mutators
- Console output
- String concatenation
- `public static void main()`

# Methods: return types and parameters

- A method is an instruction, a function, a verb
- Java methods can take input (parameters) and give output (return)
- An example could be:



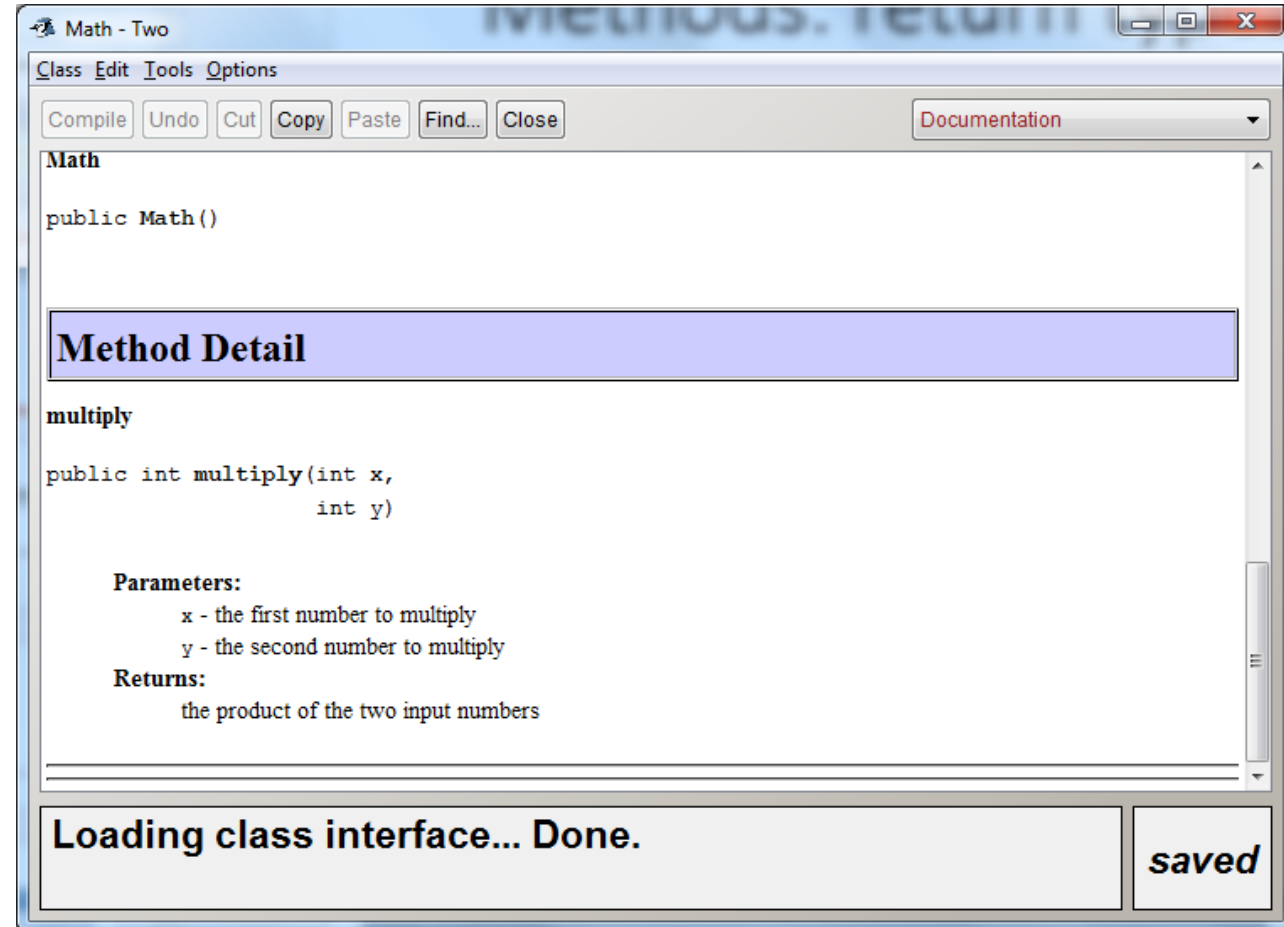
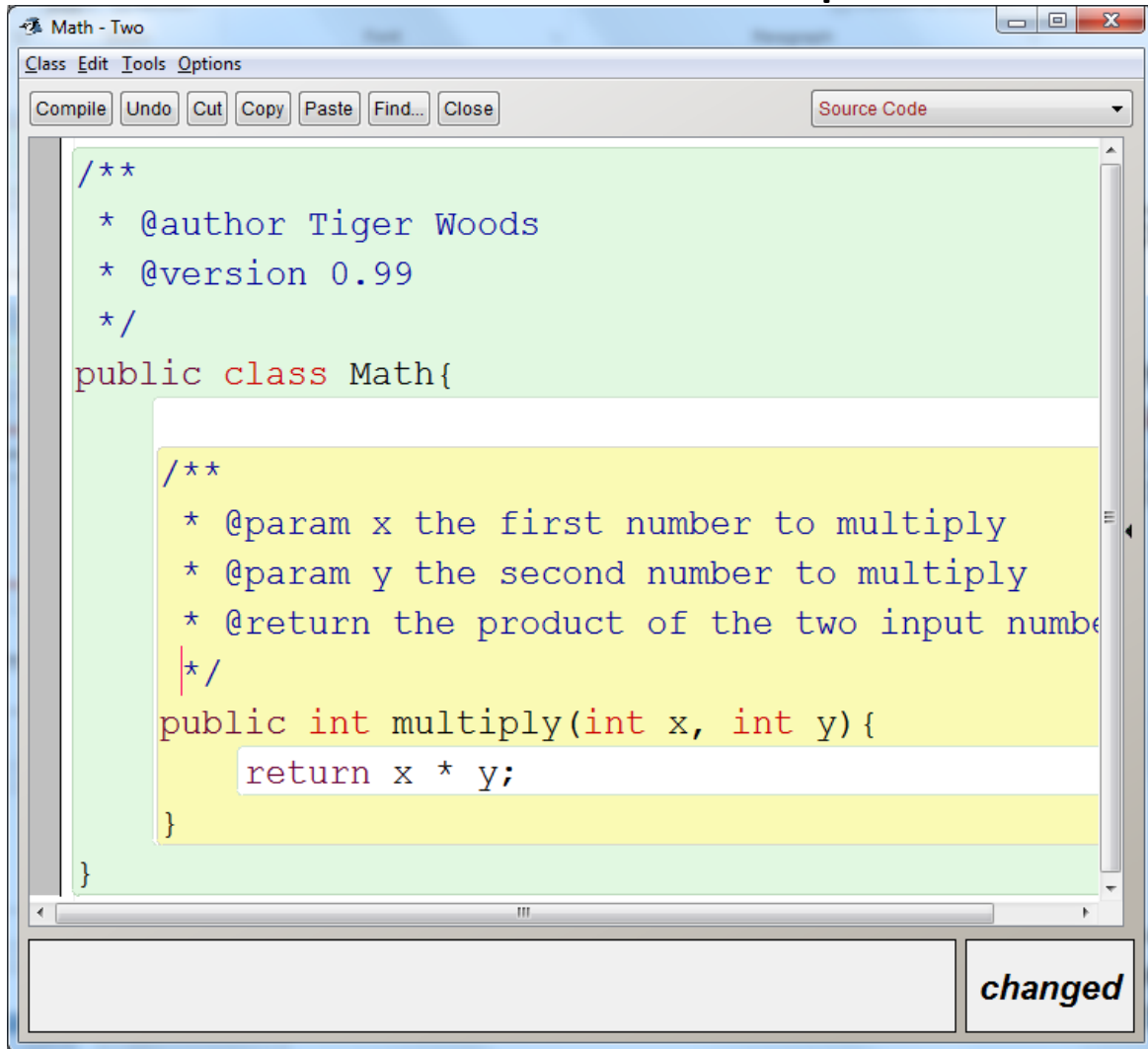
The screenshot shows a Java IDE window titled "Math - Two". The window has a menu bar with "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". A "Source Code" dropdown menu is also present. The main text area contains the following Java code:

```
public class Math{  
    public int multiply(int x, int y){  
        return x * y;  
    }  
}
```

At the bottom right of the window, there is a button labeled "changed".

# Methods: return types and parameters

- Since this method is public it needs JavaDoc comments on top of it.



# Methods: return types and parameters

- Methods should be named as verbs
- If a particular method doesn't need to return anything, its return *type* is **void**
- Example: 

```
public void returnNothing(){  
    }
```
- Notice that this method also didn't need any inputs
- Inputs are called parameters or arguments.



# Methods: return types and parameters

- A method signature includes:

**modifier return-type name(parameters)**

- Examples:

public boolean	isStudentEnrolled(String studentID)
private void	doSomething(double a, int b, char c)
public double	getAverage(double num1, double num2, double num3)
private char	doSomethingElse()

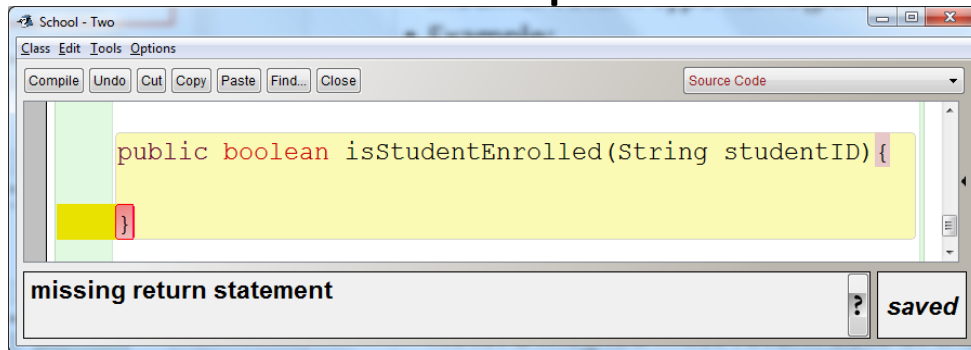
# Methods: return types and parameters

```
public boolean isStudentEnrolled(String studentID)
private void    doSomething(double a, int b, char c)
public double   getAverage(double num1, double num2, double num3)
private char     doSomethingElse()
```

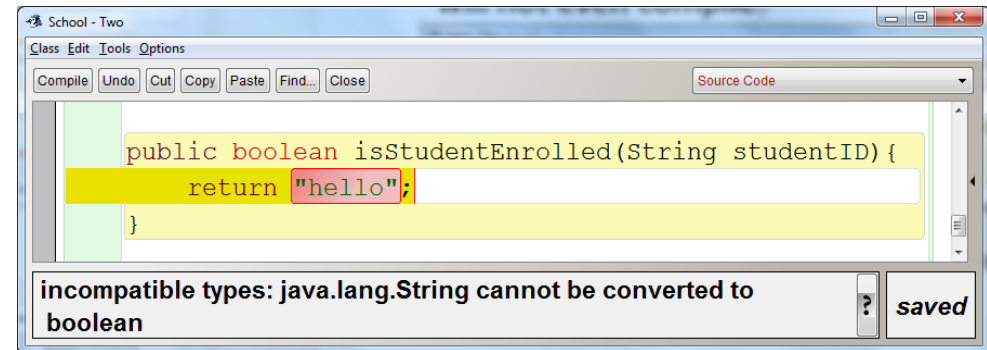
- Call those methods later:
- `isStudentEnrolled("a00123456")`
- `doSomething(5.5, 4, 'c')`
- `getAverage(1.0, 2.2, 3.4)`
- `doSomethingElse()`

# Methods: return types and parameters

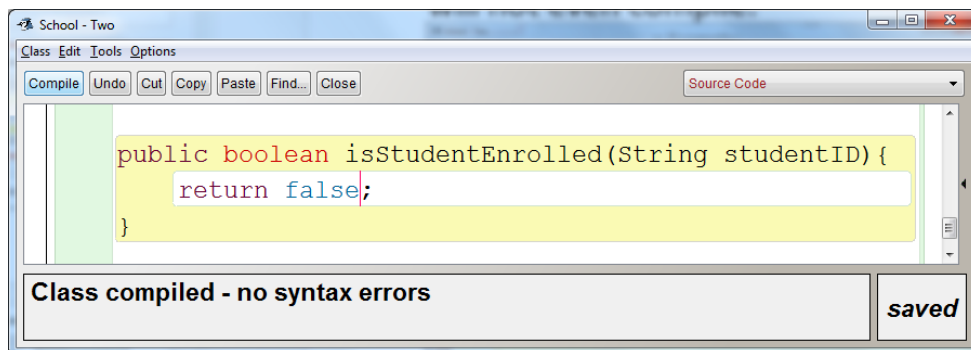
- Note: a method *must* return data of the specified return type, or it will not even compile.



A screenshot of a code editor window titled "School - Two". The code editor shows a method definition: `public boolean isStudentEnrolled(String studentID) {` followed by a closing brace `}`. The status bar at the bottom displays the message "missing return statement" and a "saved" button.



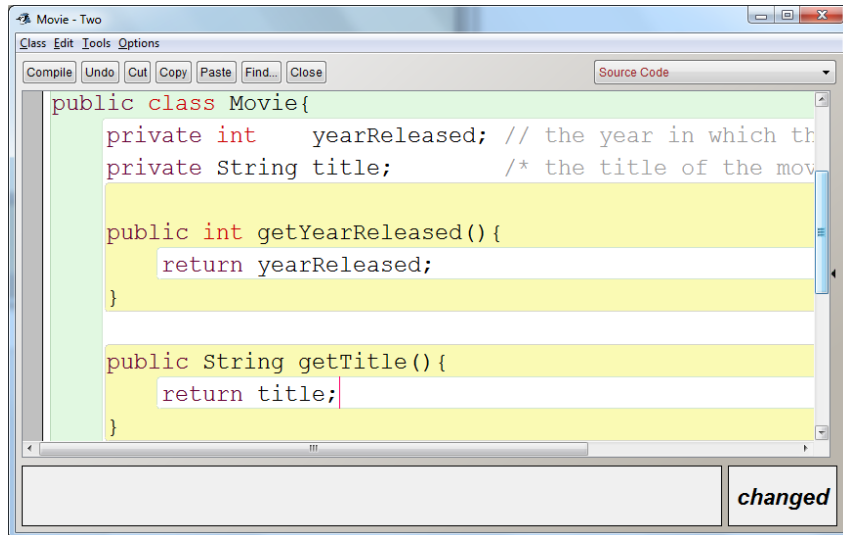
A screenshot of a code editor window titled "School - Two". The code editor shows a method definition: `public boolean isStudentEnrolled(String studentID) {` followed by `return "hello";` and a closing brace `}`. The status bar at the bottom displays the message "incompatible types: java.lang.String cannot be converted to boolean" and a "saved" button.



A screenshot of a code editor window titled "School - Two". The code editor shows a method definition: `public boolean isStudentEnrolled(String studentID) {` followed by `return false;` and a closing brace `}`. The status bar at the bottom displays the message "Class compiled - no syntax errors" and a "saved" button.

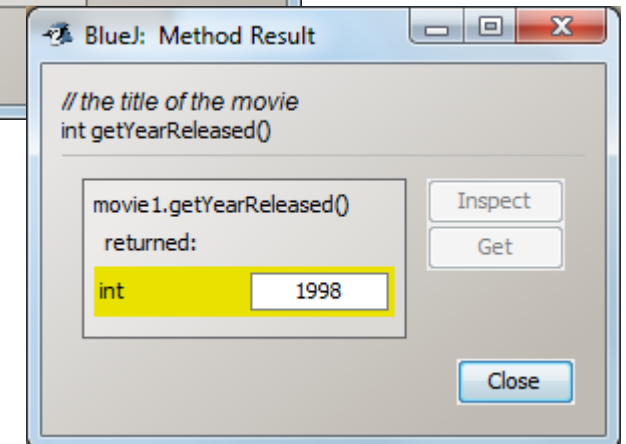
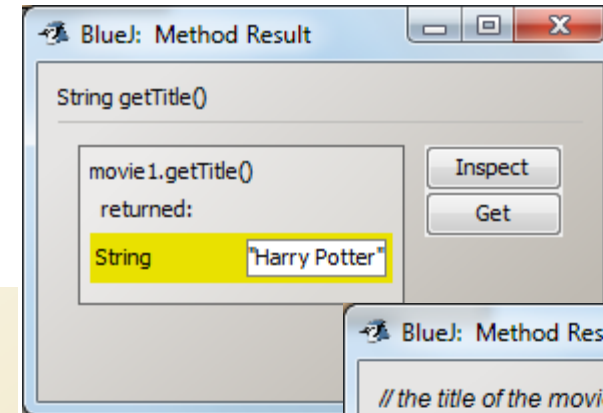
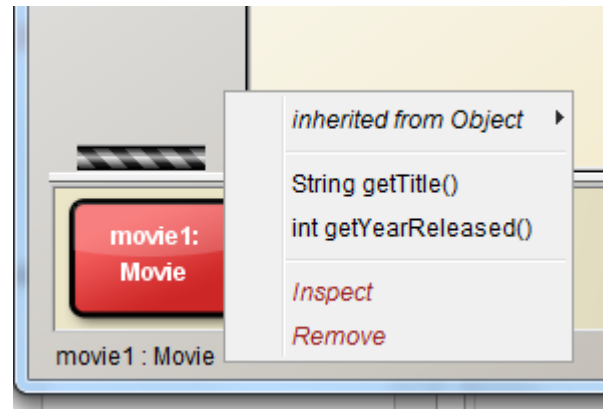
# Accessors

- Accessor methods are also known as “get methods” or “getters”
- They are only one of many “categories” of methods that we can make
- Accessor methods give other classes access to private data (instance variables, for example).



```
public class Movie{  
    private int    yearReleased; // the year in which th  
    private String title;        /* the title of the mov  
  
    public int getYearReleased(){  
        return yearReleased;  
    }  
  
    public String getTitle(){  
        return title;  
    }  
}
```

changed



# Accessors

- The common method signature for accessor methods:
- `public <return type> getInstanceVariableName()`
- Usually, no parameters are necessary
- The return type matches the type of the instance variable being returned
- Often, the accessor method does nothing except simply return data
- Otherwise the data is inaccessible outside this class, since we always make our instance variables *private*.

# void

- If a method returns nothing at all, its return type must be declared as void

```
public void doSomethingButReturnNothing(){  
    return; // not necessary, but possible  
}
```

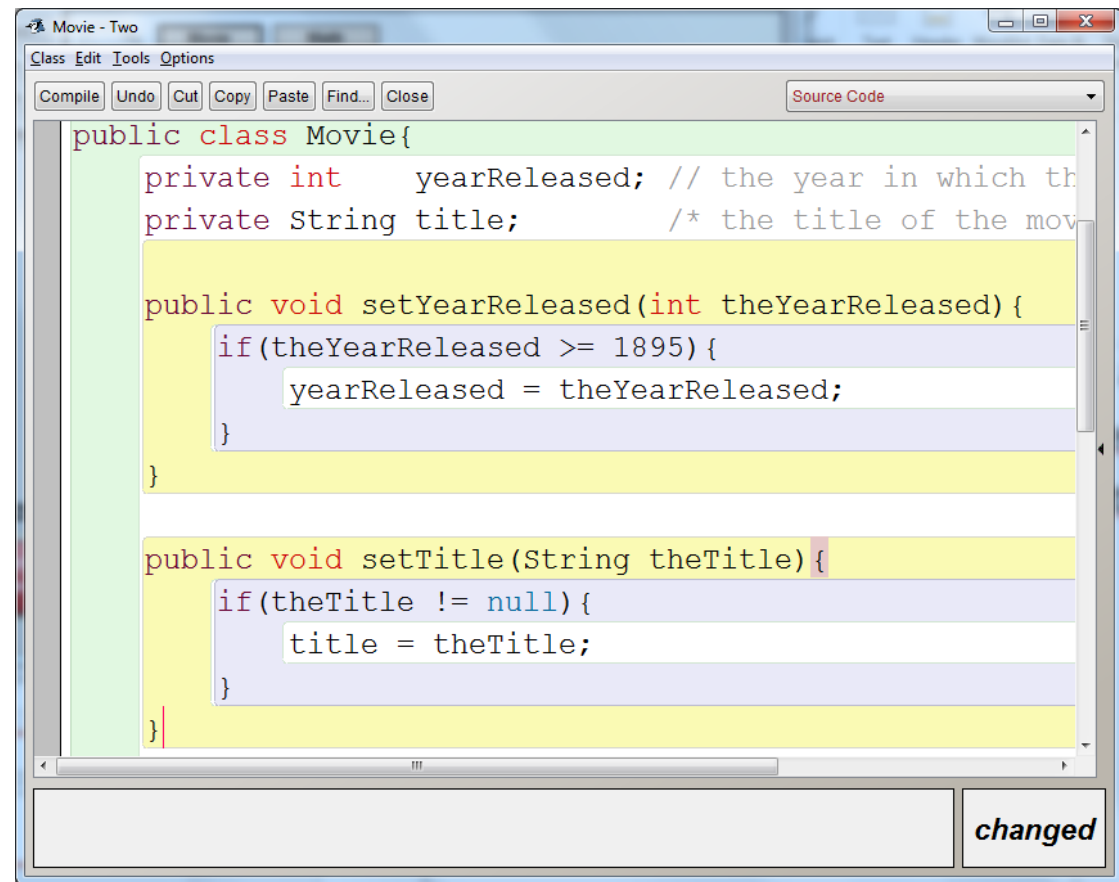
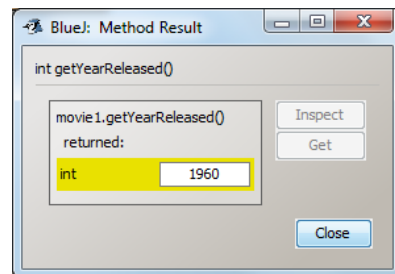
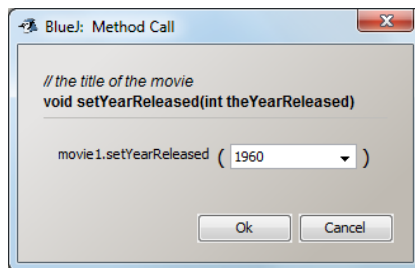
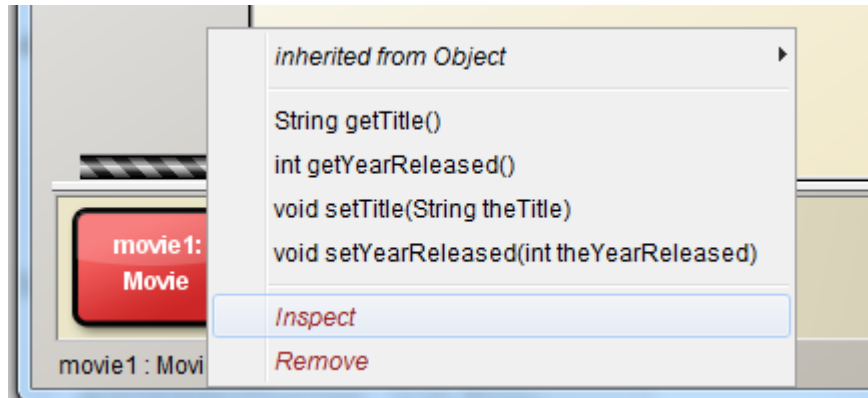
- When a method returns, it stops running (with any return type).

# Mutators

- Another category (out of many) of method is the mutator method
- Also known as “set method” or “setter”
- Allows other classes to change instance variables.
- Mutators generally have rules that must be followed first; if the rules are not followed, then the data is not changed
- These rules are what make mutators useful. We protect the data this way.
- Without any guard rules, we may as well have made the instance variables public (a bad idea since any code anywhere could change the data to any value)
- A class often has more accessors than mutators.

# Mutators

- In general, the method signature for a mutator method:
- `public void setInstanceVariableName(<data type> newVariableValue)`
- Our constructors act similarly to mutator methods, too.





# Logical operators

- `&&` `((this is bcit) && (this is sfu) && (this is MIT)) // false at sfu condition`
- `||` `((this is bcit) || (this is sfu) || (this is MIT)) // true at bcit condition`
- `!` `boolean open = true; open = !open; x != 5`
- Short circuiting

# Methods

- Many methods are public
- However, if your method is only intended to be used inside its own class, make that method private instead
- Accessors and mutators are only two of many “types” of methods.

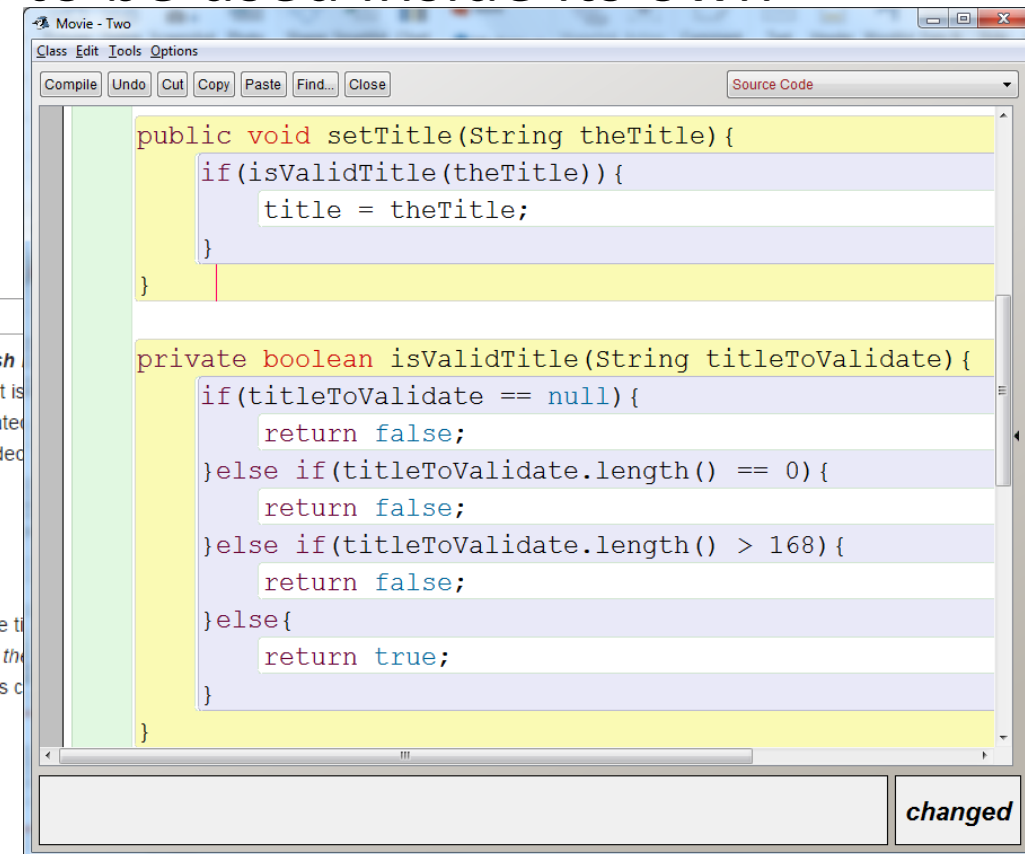
## Part 2 [\[edit\]](#)

Titled *Night of the Day of the Dawn of the Son of the Bride of the Return of the Revenge of the Terror of the Attack of the Evil, Mutant, Alien, Flesh Eating, Hellbound, Zombified Living Dead Part 2* sometimes with the added title *in Shocking 2-D* is a 1991 horror spoof written and directed by James Riffel.<sup>[2]</sup> It is known as *NOTDOT*. Although dubbed Part 2, it is the first part actually released publicly. The movie where he used an alias name, **Lowell Mason**, was created by redubbing the 1968 horror classic *Night of the Living Dead* with comedic dialog, and by adding new clips. Although it was originally distributed to only 500 video stores in the United States, the film has since achieved cult status.<sup>[citation needed]</sup>

*NOTDOT* was screened at the New York City Horror Film Festival in October 2005.<sup>[3]</sup>

## Title [\[edit\]](#)

With 41 words in its title, 168 characters without spaces, it holds the distinction of being cited as the movie with the longest English language title,<sup>[4]</sup> but of the title *Night of the Day of the Dawn of the Son of the Bride of the Return of the Revenge of the Terror of the Attack of the Evil, Mutant, Alien, Flesh Eating, Hellbound, Zombified Living Dead Part 2: In Shocking 2-D* (1991) as the longest English-language title of all time, but it's considered a gimmicky joke.<sup>[5]</sup>



```
Movie - Two
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Close Source Code

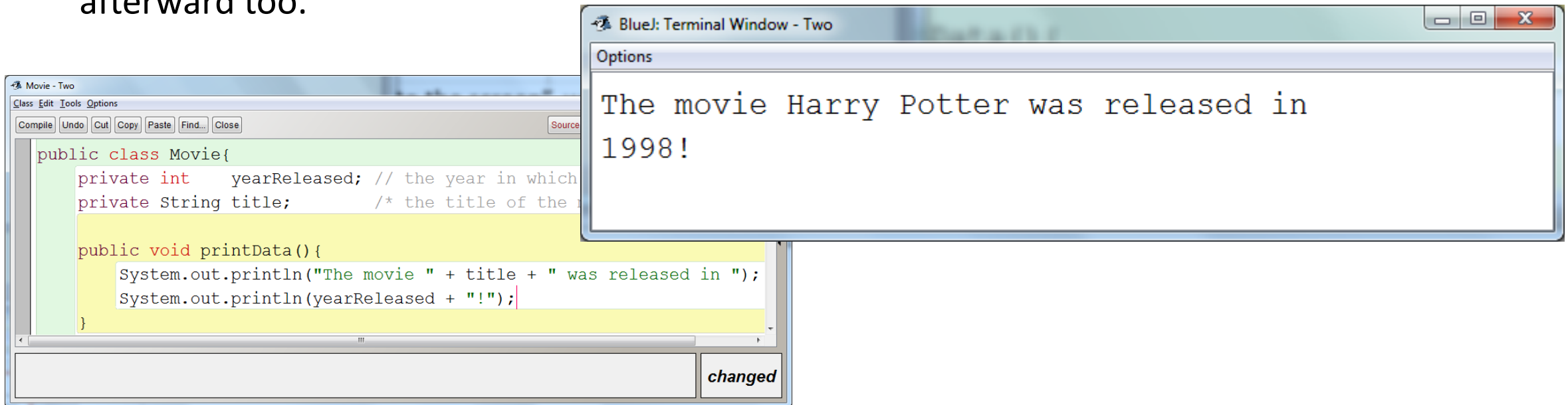
public void setTitle(String theTitle){
    if(isValidTitle(theTitle)){
        title = theTitle;
    }
}

private boolean isValidTitle(String titleToValidate){
    if(titleToValidate == null){
        return false;
    }else if(titleToValidate.length() == 0){
        return false;
    }else if(titleToValidate.length() > 168){
        return false;
    }else{
        return true;
    }
}
```

changed

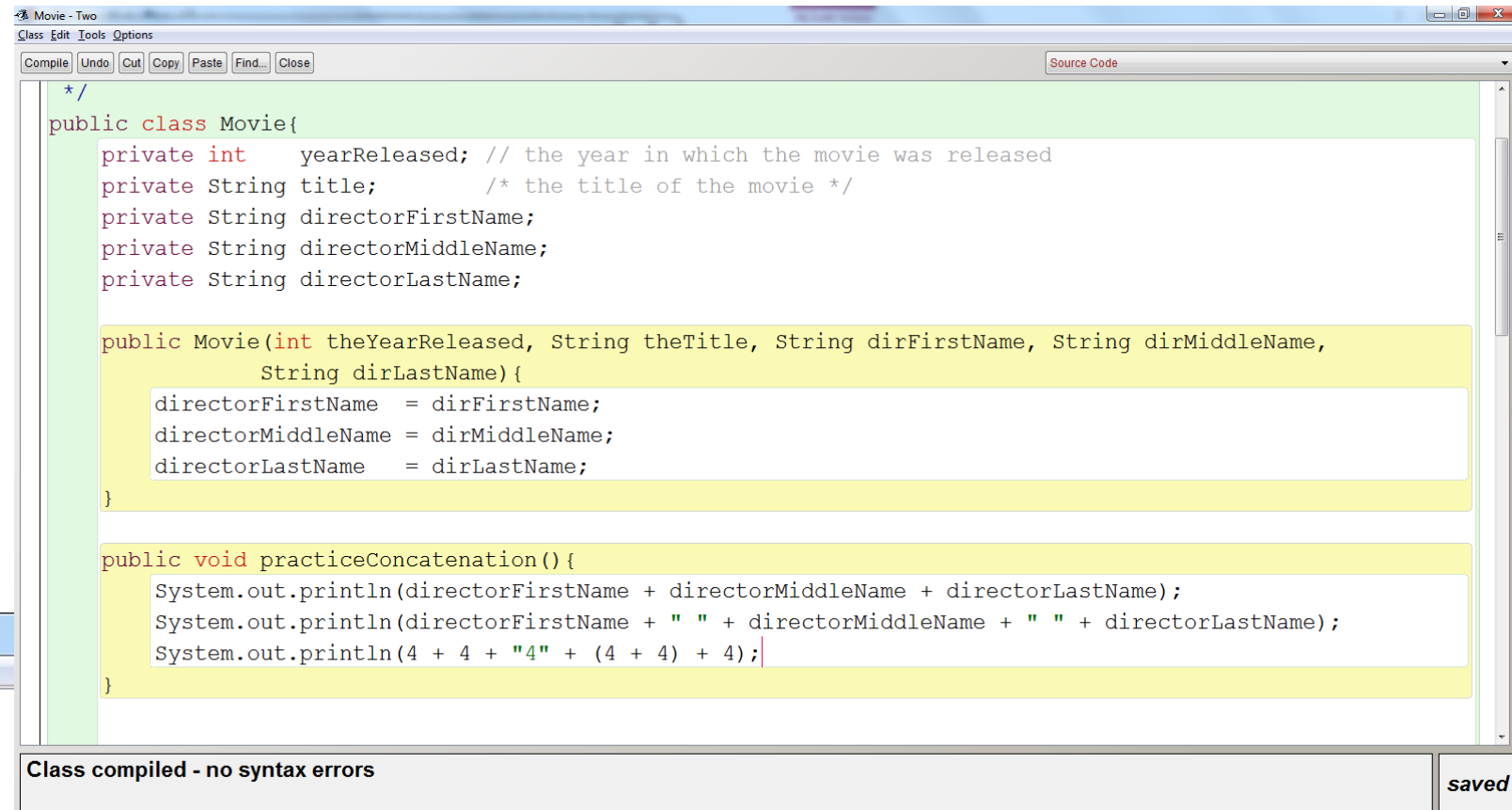
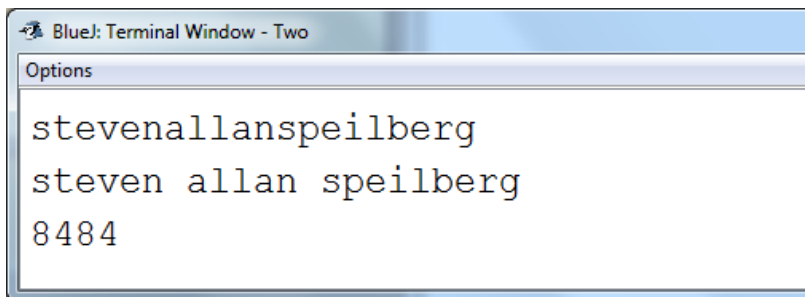
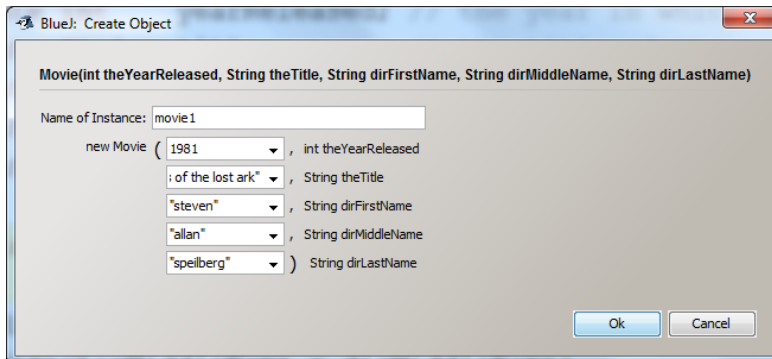
# Console output

- If we want to “write to the screen”, we will do what is better known as “output to the console”
- We can use the `System.out.print()` method to write something to the screen
- `System.out.println()` does the same thing but then writes a “new line” character afterward too.



# String concatenation

- Concatenation means “join”
- The plus sign + means addition when numbers are involved, but means “join” when Strings are involved.

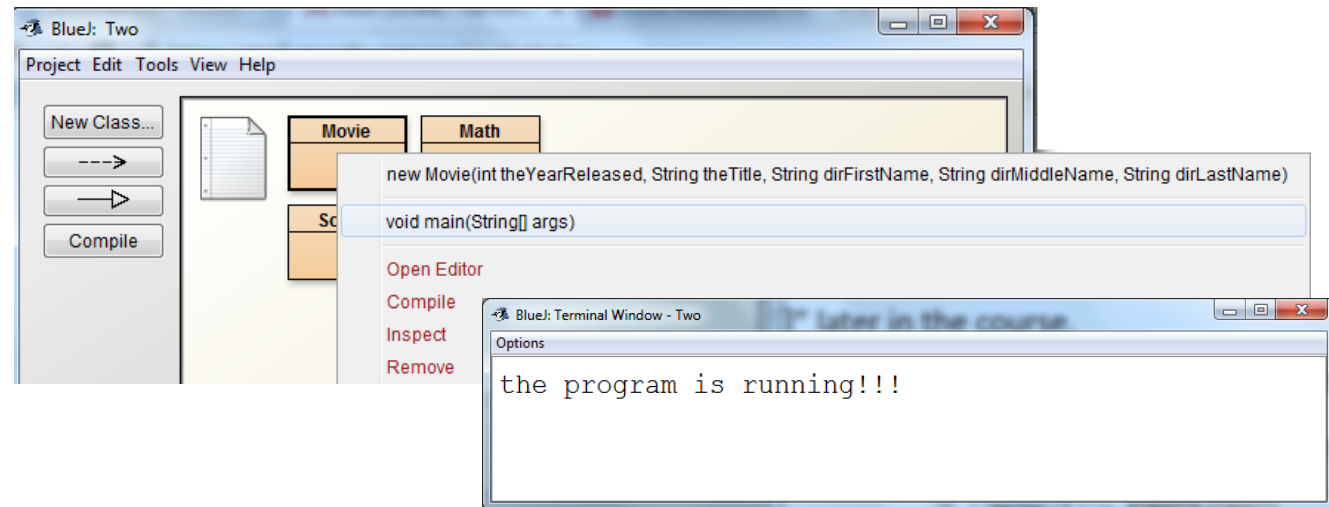
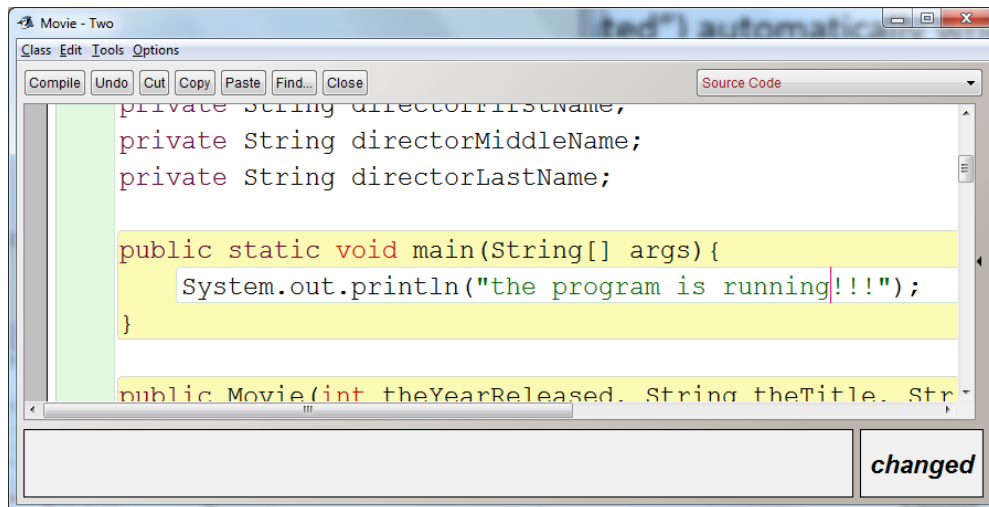


# String concatenation

- `System.out.println(5 + 5 + "5" + (5 + 5) + 5 + 5 + "(5+5)");`
- `1051055(5+5)`
- `System.out.println(5 + 5 + (5 + 5) + 5 + 5 + "(5+5)");`
- `30(5+5)`

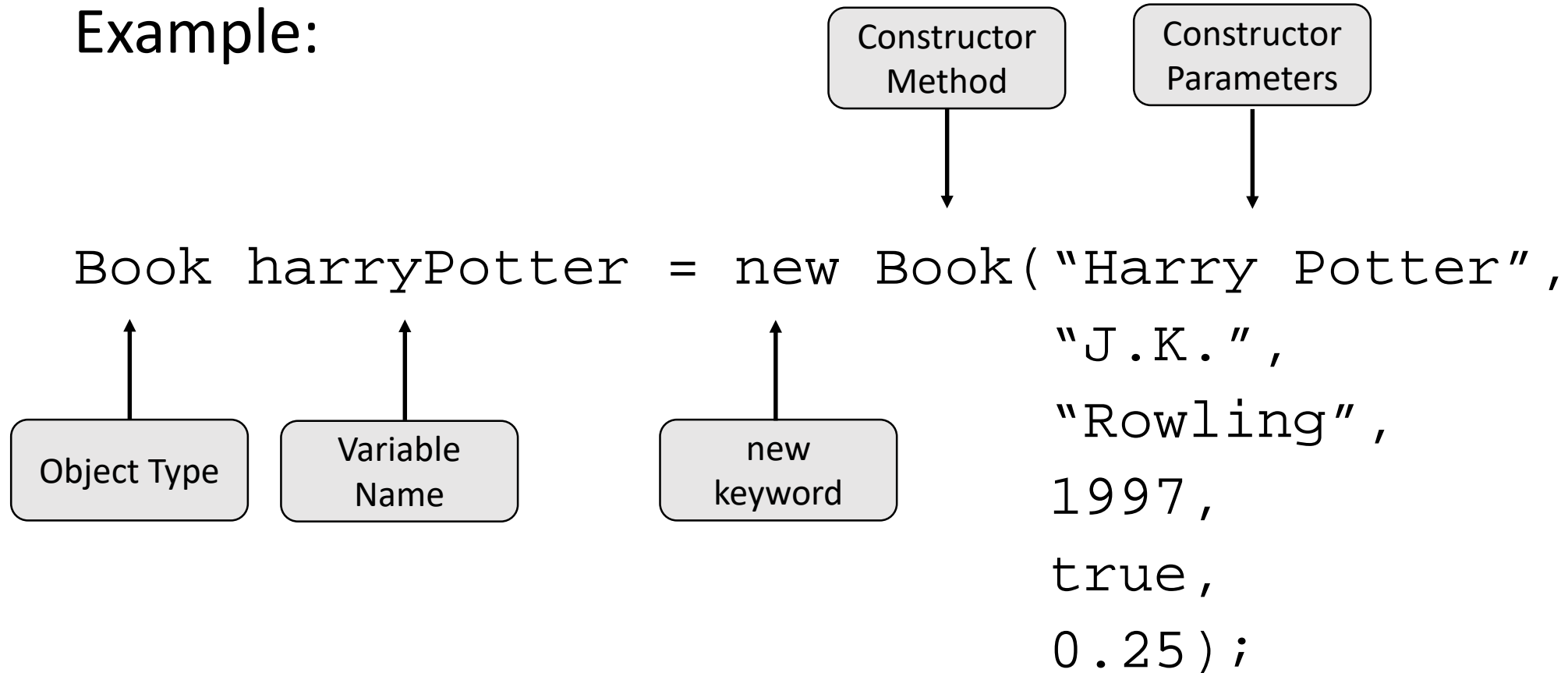
# public static void main(String[] args)

- This exact method signature inside of a class means this method could be run (“executed”) automatically when you make a program.
- Until this point we have been making data classes.
- With the inclusion of public static void main, we now have an executable *program*.
- More about “static” and “[ ]” later in the course.
- Run this method from its *class* in BlueJ (as opposed to from an *object*).



# Creating an Instance of a Class in Java

Example:



# Creating Multiple Instances of a Class in Java

```
Book harryPotter = new Book("Harry Potter",  
                             "J.K.",  
                             "Rowling",  
                             1997,  
                             true,  
                             0.25);
```

```
Book introToJava = new Book("Introduction to Java Programming",  
                             "Daniel",  
                             "Liang",  
                             2013,  
                             false,  
                             0.75);
```



# Now We Can Call Methods!

```
Book harryPotter = new Book("Harry Potter",  
                             "J.K.",  
                             "Rowling",  
                             1997,  
                             true,  
                             0.25);
```

```
System.out.println("Title: " + harryPotter.getTitle());  
System.out.println("Author First Name: " + harryPotter.getFirstName());  
System.out.println("Author Last Name: " + harryPotter.getLastName());
```

# Some Review Questions...

- What are the inputs of a method called? The output?
- What are two types of methods? What is the purpose of each?
- What does void mean in the context of a method?

# Some Review Questions...

- How do we print output in Java?
- What is the operator to concatenate Strings?

# Some Review Questions...

When do the following logical expressions short circuit?

```
int a = 1  
int b = 2  
int c = 3
```

```
if ((a == 2) || (b == 2) || (c == 2)) {  
    // b == 2  
}
```

```
if ((a == 1) && (b == 1) && (c == 1)) {  
    // b == 1  
}
```

# Pre-Lab Demo

- Sample Code
  - MobilePhone.java
    - Runnable static main method
  - Main.java
    - Creates MobilePhone objects