Ticket Machine

Variables, Parameters, Operators, Comments

Produced by: Dr. Siobhán Drohan

(based on Chapter 2, Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling)

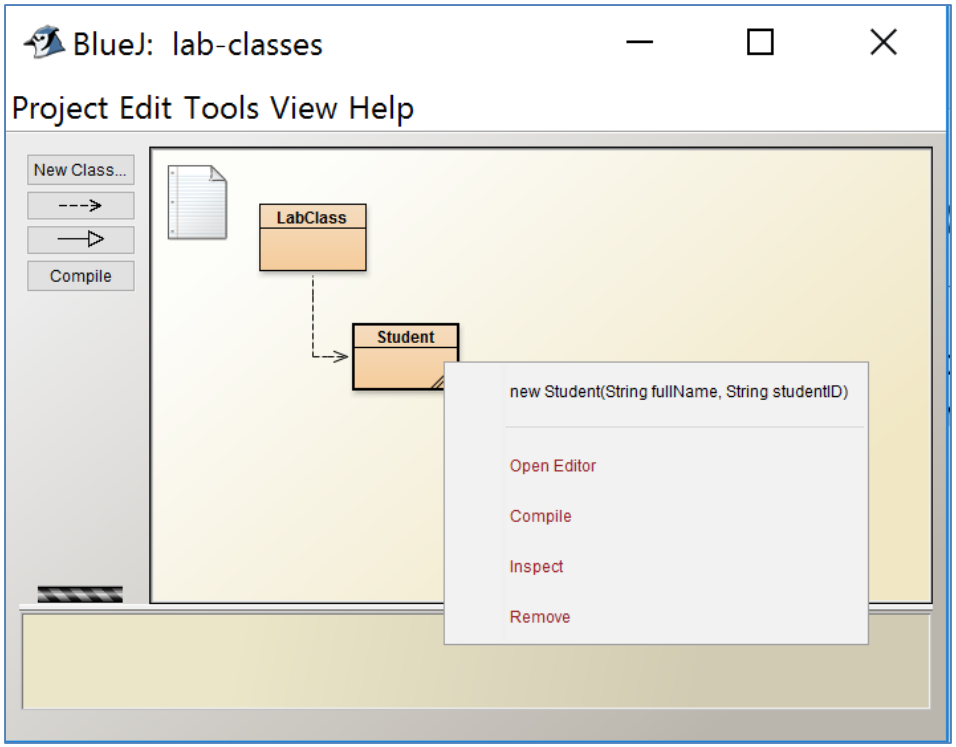
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

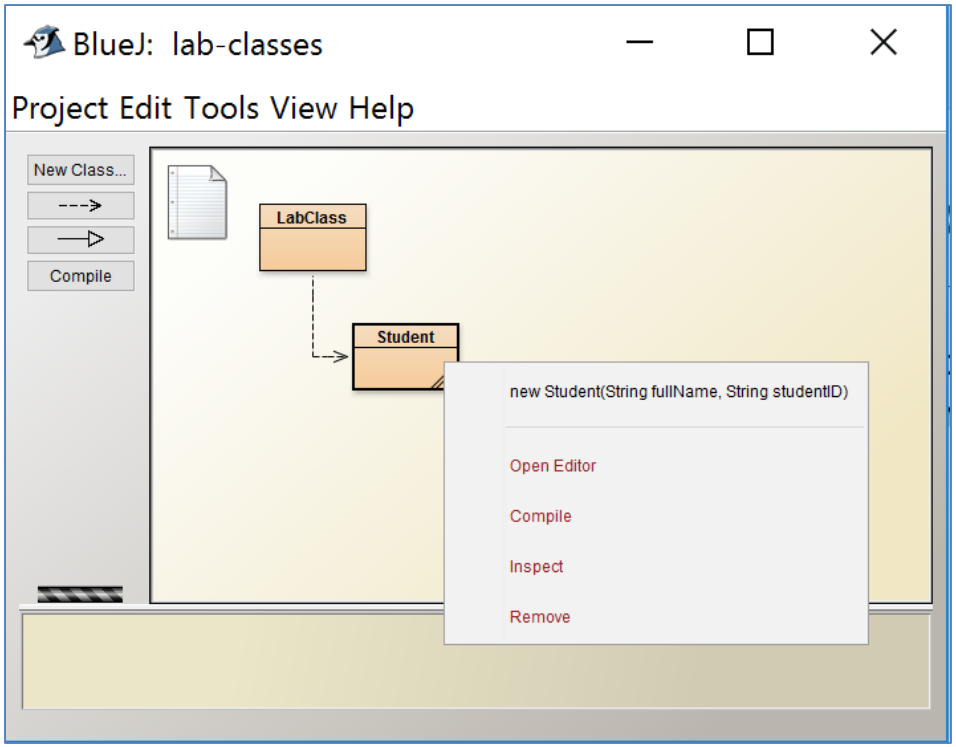
Recap: Constructors with parameters



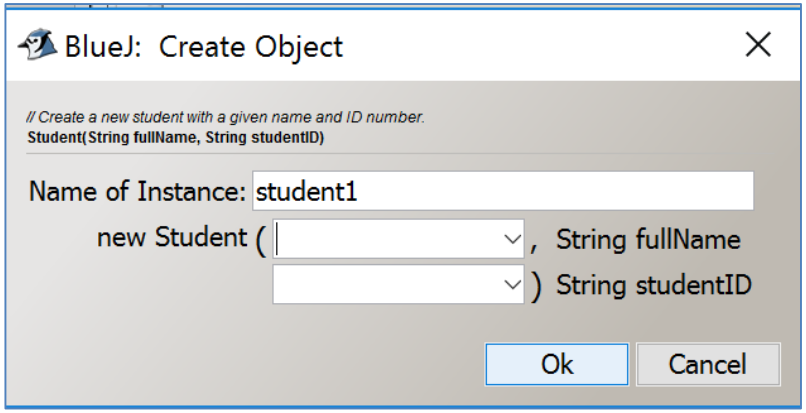
Recap:

- A constructor is a special method that is the same name as the class.
- It “constructs” the object i.e. creates an instance of the class.

Recap: Constructors with parameters

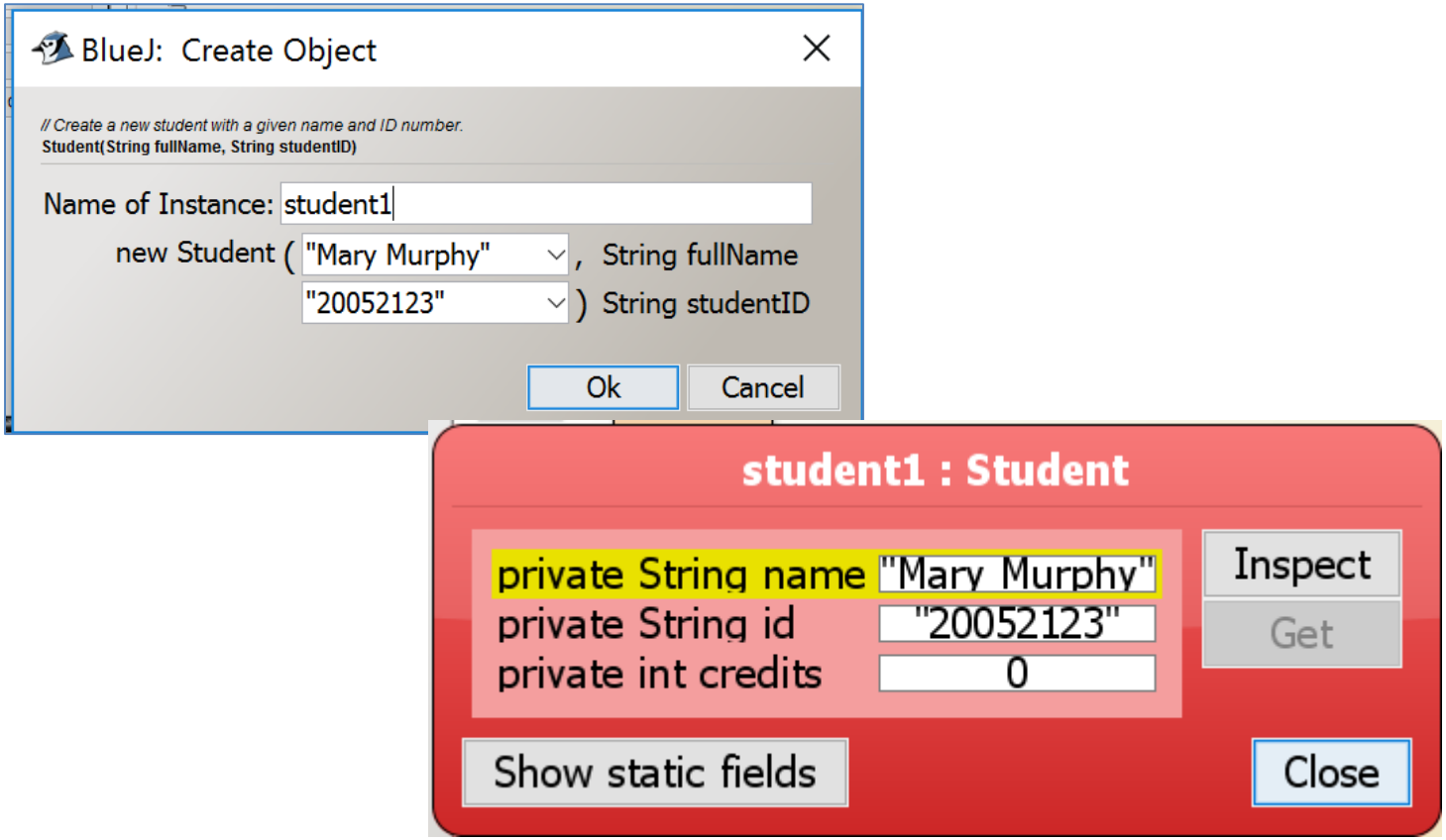


When a constructor with parameters is called, a window will pop up asking you to enter the required information:



Recap: Constructors with parameters

The entered information is then used to set up the starting state of the object:



Recap: Constructors with parameters

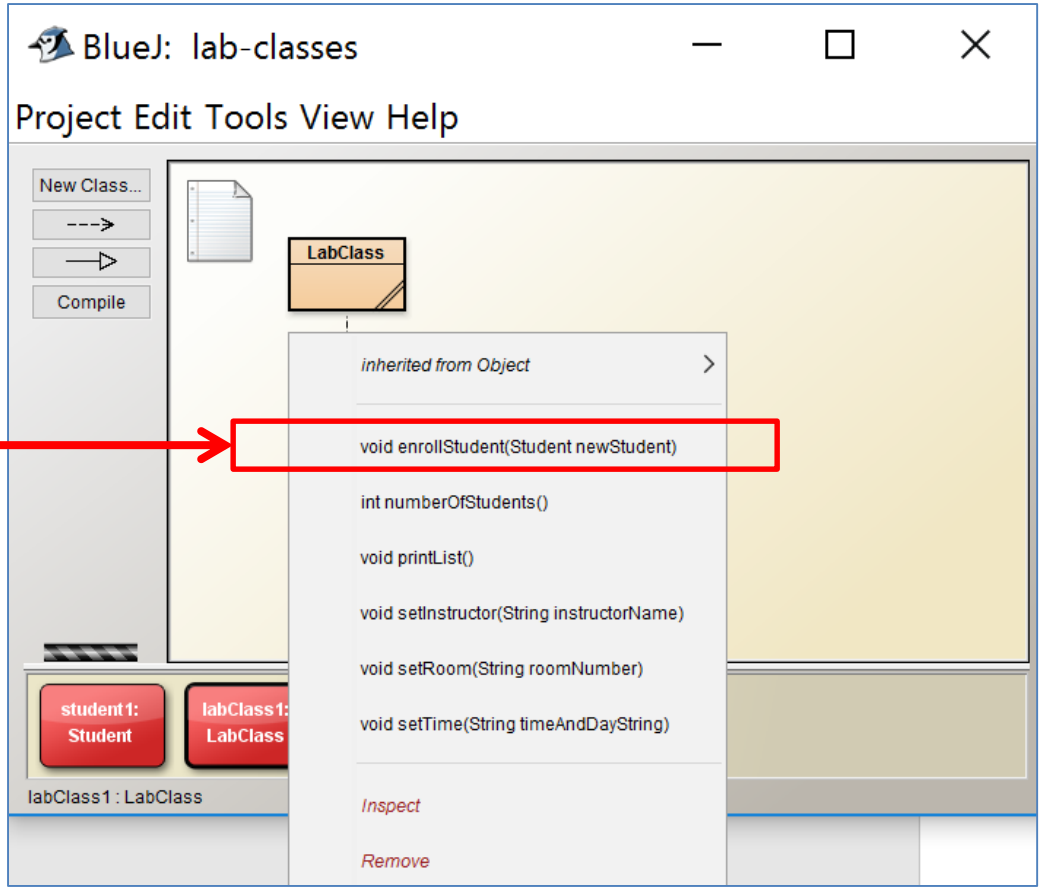
A constructor typically sets a starting state for an object.

Student.java

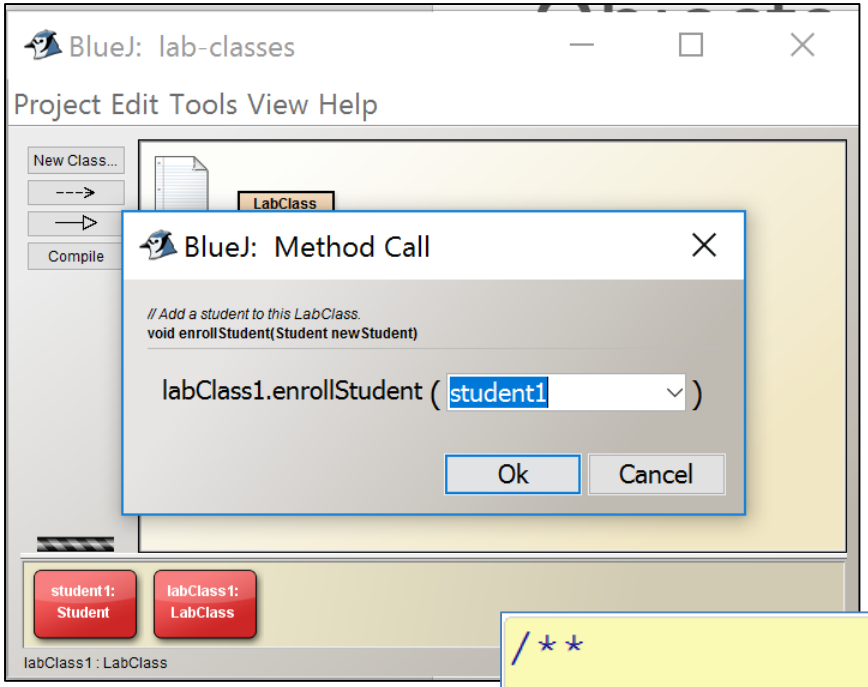
```
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String fullName, String studentID)
    {
        name = fullName;
        id = studentID;
        credits = 0;
    }
```

Recap: Objects as parameters

- Objects can be passed as parameters to methods of other objects.

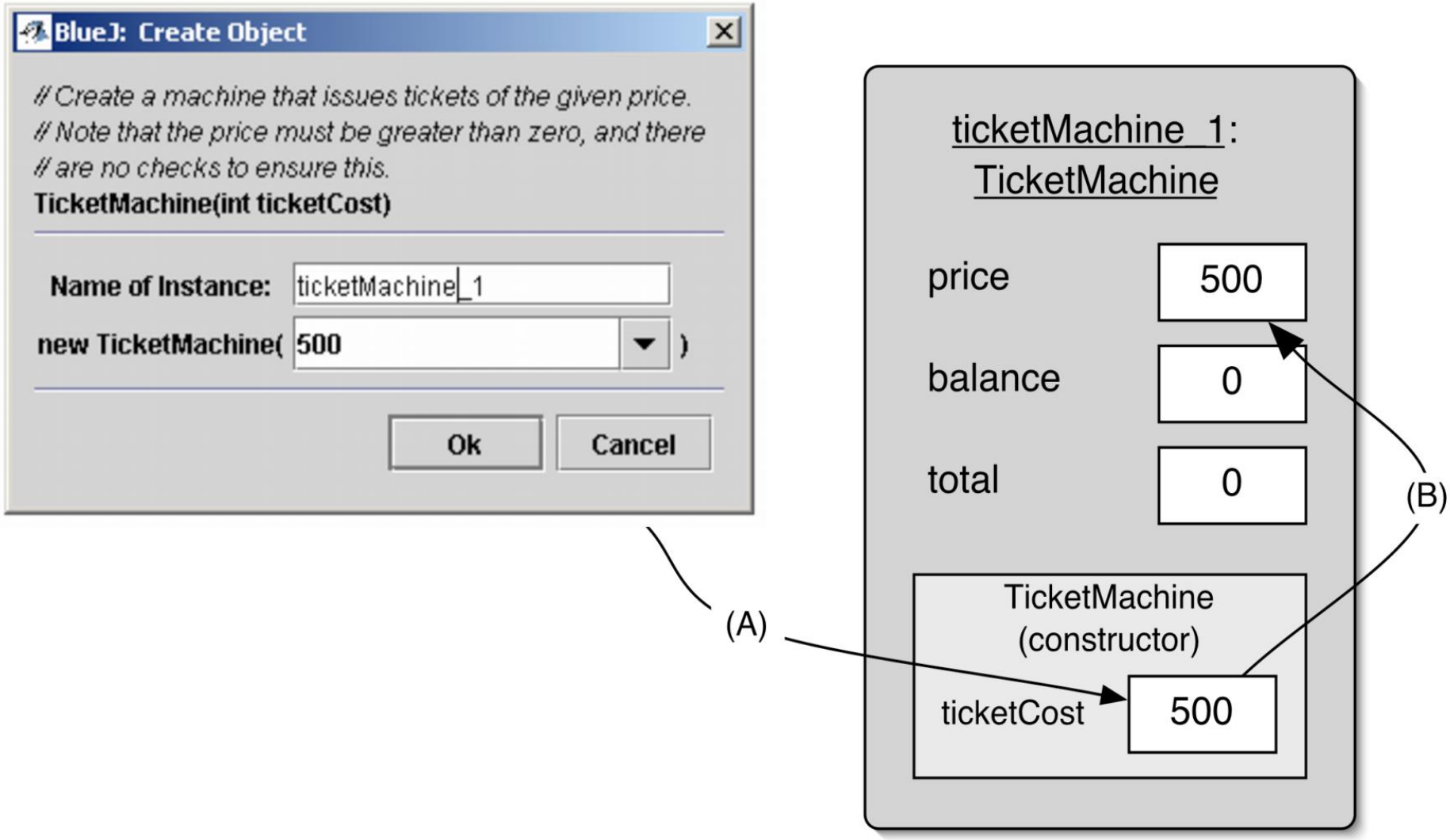


Recap: Objects as parameters




```
/**
 * Add a student to this LabClass.
 */
public void enrollStudent(Student newStudent)
{
    //code ommitted
}
```

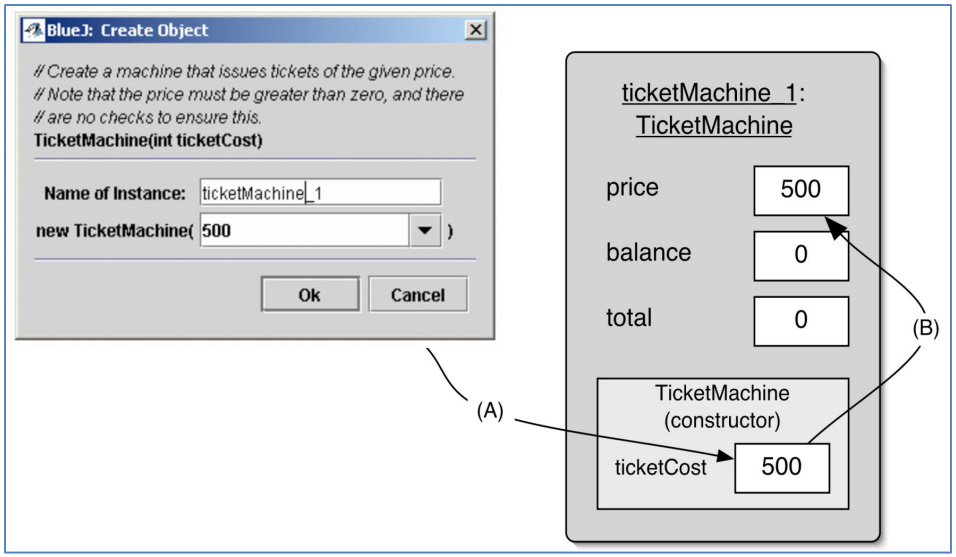
Naïve Ticket Machine: passing data via parameters



Parameters

- Variable names are the *formal parameters* e.g. ***ticketCost***

- The values are the *actual parameters* e.g. user-supplied value, such as ***500***, is an actual parameter. Note: actual parameters are also called *arguments*.




```
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Printing from methods

Naïve ticket machine

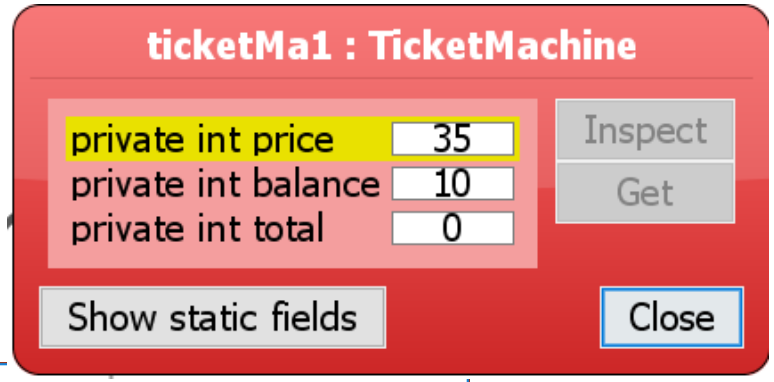
```java
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

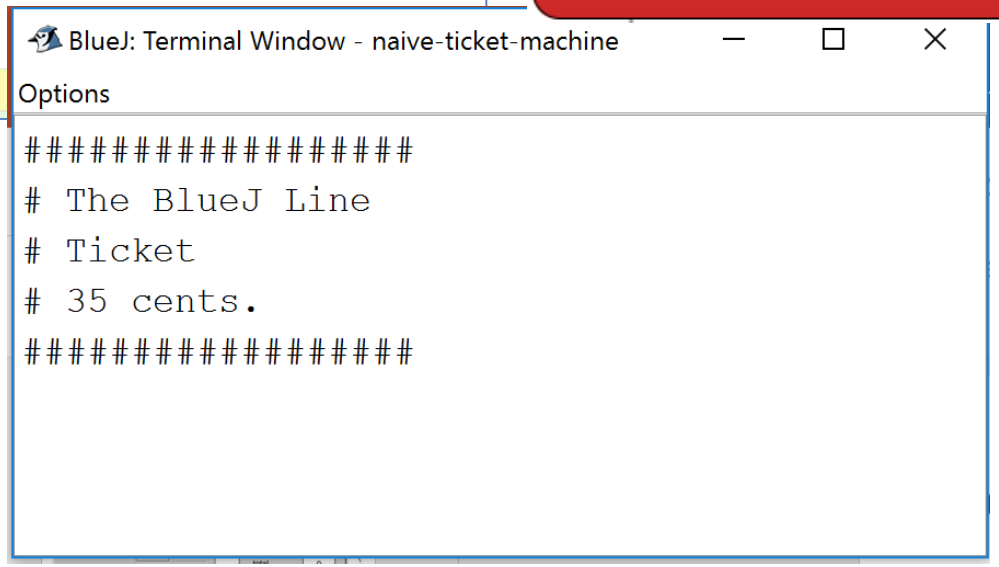
```java
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
```

Printing from methods

Naïve ticket machine

```java
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```





Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

+ Operator

Naïve ticket machine

```java
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

+ operator used to concatenate Strings

+ operator used as an addition operator

```java
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
```

+ Operator

Naïve ticket machine

```
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

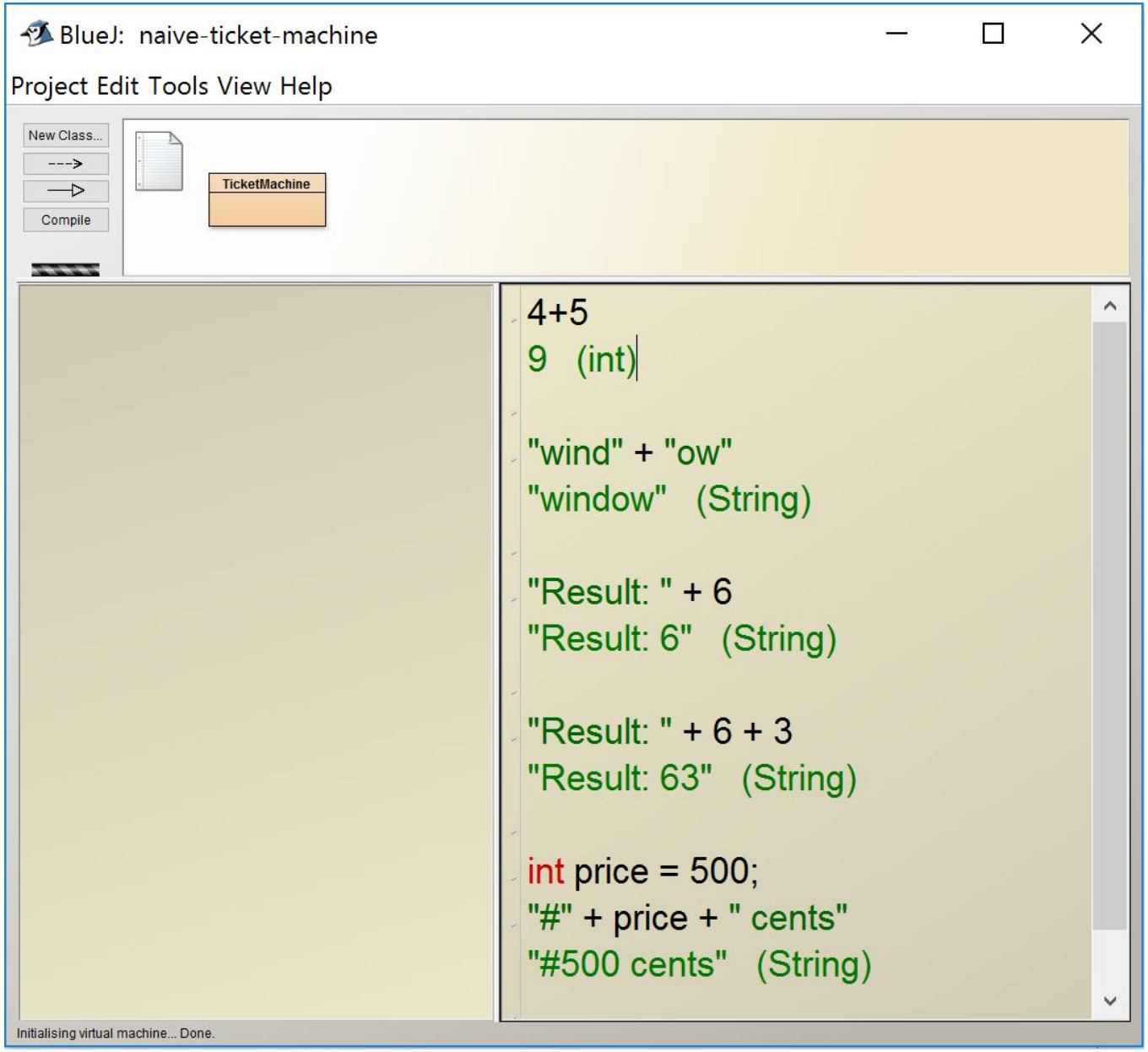
When used between a String and anything else, '+' is a string-concatenation operator i.e. it concatenates or joins Strings together to create a new String.

When used between two numeric types, it is an arithmetic addition operator.

→ Operator overloading

+ Operator examples

(BlueJ CodePad)



Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Recap: reflecting on the ticket machines

- The naïve-ticket-machine behavior is inadequate in several ways:
  - No checks on the amounts entered.
  - No refunds.
  - No checks for a sensible initialization.

- How can we do better?
  - We need more sophisticated behavior.

demo

better ticket machine

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Variables

In Programming, variables:

- are created (defined) in your programs.
- are used to store data (whose value can change over time).
- have a data type.
- have a name.
- are a VERY important programming concept.

Variable names…

- Are case-sensitive.
- Begin with either:
  - a **letter (preferable)**,
  - the dollar sign "$", or
  - the underscore character "_".
- Can contain letters, digits, dollar signs, or underscore characters.
- Can be any length you choose.
- Must not be a **keyword or reserved word** e.g. int, while, etc.
- Cannot contain white spaces.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Variable names should be carefully chosen

- Use full words instead of cryptic abbreviations e.g.
  - variables named speed and gear are much more intuitive than abbreviated versions, such as s and g.

- If the name consists of:
  - only one word, spell that word in all lowercase letters e.g. ratio.
  - more than one word, capitalise the first letter of each subsequent word e.g. gearRatio and currentGear.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html

Variable Scope: Global

- Instance fields are one sort of variable:
  - They store values through the life of an object.
  - They are accessible throughout the class (i.e. global).
  - They are defined at the top of the class.

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

Variable Scope: Local

- Constructors and methods can include shorter-lived variables:
  - They exist only as long as the constructor/method is being executed.
  - They are only accessible from within the constructor/method (i.e. local).

```
public class ClassName
{
    //Instance Fields
    //Constructors
    //Methods
}
```

Variable Scope

Better ticket machine

```
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
```

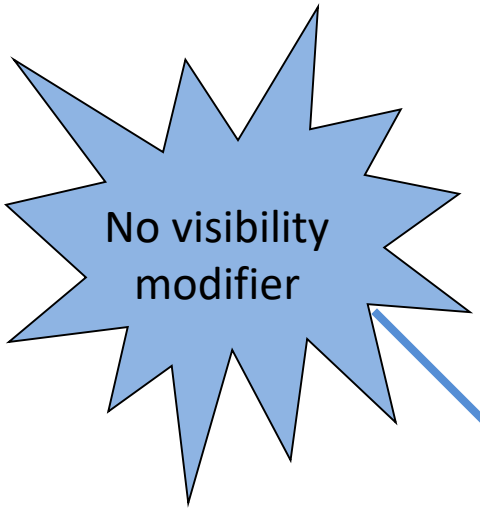
Three Global Variables

```
/**
 * Return the money in the balance.
 * The balance is cleared.
 */
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

Local Variable

Variable Scope

Better ticket machine




```
/**
 * Return the money in the balance.
 * The balance is cleared.
 */
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

Local Variable

Variable Scope

Better ticket machine

Local Variable

```
/**
 * Create a machine that issues tickets of the given price.
 */
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

Scope and lifetime

- The scope of a local variable is the block it is declared in i.e. the { }

- The lifetime of a local variable is the time of execution of the block it is declared in i.e. the length of time it takes for the method to run.

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

this keyword

Instance fields – global scope

```
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String fullName, String studentID)
    {
        name = fullName;
        id = studentID;
        credits = 0;
    }
```

Local variables

Student.java

this keyword

What if we wanted to call our local variables the same name as our instance fields?

Would this work?

Instance fields – global scope

```
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String name, String id)
    {
        name = name;
        id = id;
        credits = 0;
    }
```

Local variables

Student.java

this keyword

```java
public class Student
{
    // the student's full name
    private String name;
    // the student ID
    private String id;
    // the amount of credits for study taken so far
    private int credits;

    /**
     * Create a new student with a given name and ID number.
     */
    public Student(String name, String id)
    {
        this.name = name;
        this.id = id;
        credits = 0;
    }
```

Answer is NO. We need to use the “this” keyword.

this keyword – the theory!

- The class Student contains three fields:
  - name, id, credits
- The Student constructor takes two parameters, also named:
  - name, id
- This is called variable name overloading. When we refer to the name variable, how does Java know which variable we mean?
- We need a way to distinguish between them!

this keyword – the theory!

- We can use the this keyword to distinguish between them.
- The expression this refers to the current object.
- In the Student constructor, writing:
  - this.name refers to the name field in the current object.
  - name refers to the name field in the parameter list.

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Boolean conditions

- A boolean condition is an expression that evaluates to either true or false e.g.

```
public void insertMoney(int amount){
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " +
                           amount);
    }
}
```

- An if statement evaluates a boolean condition and its result will determine which portion of the if statement is executed.

Operators: Arithmetic

Arithmetic Operator	Explanation	Example(s)
+	Addition	6 + 2 amountOwed + 10
-	Subtraction	6 – 2 amountOwed – 10
*	Multiplication	6 * 2 amountOwed * 10
/	Division	6 / 2 amountOwed / 10

Operators: Relational

Operator	Use	Returns true if
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than to op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

Source: http://www.freejavaguide.com/relational_operators.htm

Operators: examples

Better ticket machine

```java
public void printTicket(){
    if(balance >= price) {
        // Simulate the printing of a ticket.
        System.out.println("##################");
        System.out.println("# The BlueJ Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("##################");
        System.out.println();

        // Update the total collected with the price.
        total = total + price;
        // Reduce the balance by the prince.
        balance = balance - price;
    }
    else {
        System.out.println("You must insert at least: " +
                           (price - balance) + " more cents.");

    }
}
```

```java
public void insertMoney(int amount){
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: " +
                           amount);
    }
}
```

Operators: Logical

- Logic operators operate on boolean values.
- They produce a new boolean value as a result.
- The ones that we will use are:

&&	(and)
||	(or)
!	(not)

Operators: Logical

a && b *(and)*

- This evaluates to true if both ***a*** and ***b*** are true.
- It is false in all other cases.

a || b *(or)*

- This evaluates to true if either ***a*** or ***b*** or both are true, and false if they are both false.

!a *(not)*

- This evaluates to true of ***a*** is false, and false if ***a*** is true.

Operators: Logical - quiz

```
int a = 5;
int b = 10;
int c = 7;
```

What is the result of each of these boolean expressions:

```
(a > b) && (a < c)
(a < b) || (c < a)
!(b < a) && (c > b)
```

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Operators: Compound Assignment

```
balance += amount;
```

is shorthand for

```
balance = balance + amount;
```

```
balance -= amount;
```

is shorthand for

```
balance = balance - amount;
```

Topic List

- Parameters:
  - formal
  - actual
- Printing from methods
- + Operator (and overloading)
- Recap demo: better ticket machine
- Variables: naming and scope
- *this* keyword
- Operators: Arithmetic, Relational and Logical
- Operators: Compound Assignment
- Commenting your code

Commenting your code

- Comments are lines of text added to source code to provide explanations to human readers e.g.

```
// The price of a ticket from this machine.
private int price;
```

- Comments have no effect on the functionality of a class.

Commenting your code

```java
/**
 * TicketMachine models a naive ticket machine that issues
 * flat-fare tickets.
 * The price of a ticket is specified via the constructor.
 * It is a naive machine in the sense that it trusts its users
 * to insert enough money before trying to print a ticket.
 * It also assumes that users enter sensible amounts.
 *
 * @author David J. Barnes and Michael Kolling
 * @version 2006.03.30
 */
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;

    /**
     * Create a machine that issues tickets of the given price.
     * Note that the price must be greater than zero, and there
     * are no checks to ensure this.
     */
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
```

A single-line comment is introduced by the two characters '//', which are written with no spaces between them.

Comments beginning with '/**' and ending with '*/' are called Javadoc comments and we will discuss these in a later lecture.

Commenting your code

```java
/**
 * Print a ticket.
 * Update the total collected and
 * reduce the balance to zero.
 */
public void printTicket()
{
    /* Simulate the printing of a ticket.
     * There is no validation in this method; a ticket
     * can print be printed even if sufficient funds have
     * not been entered
     */
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

More detailed comments, often spanning several lines, are usually written in the form of multi-line comments. These start with the character pair '/*' and end with the pair '*/'.

Questions?

Study aid: Can you answer these questions?

- What is the purpose of parameters in Constructors?
- What is meant by passing Objects as parameters?
- What are formal parameters?
- What are actual parameters/arguments?
- What does String concatenation mean?
- What does operator overloading mean? Can you name two uses of the + operator?
- What is meant by variable scope?
- What is meant by the lifetime of a variable?
- What is the accepted naming convention for Java Variables?

Study aid: Can you answer these questions?

- How do you print to the console?
- What is the *this* keyword used for?
- What is the purpose of comments in source code?
- How do you write a single-line comment?
- How do you write a multi-line comment?
- How do you write a Javadoc comment?

Study aid: Can you answer these questions?

- What are
  - Arithmetic operators?
  - Relational operators?
  - Logical operators?
  - Compound Assignment operators?

- Can you write a Java code fragments using these operators?

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/