CO452
Programming Concepts

Lecture 8

Main, User Interfaces and Input

Starting our programs

So far in BlueJ, we’ve created objects on the object bench and called their methods to run our programs.

However, our programs are dependent upon at least one instantiation of a class within the BlueJ environment. We have not yet coded a way to run our programs outside the BlueJ environment.

There is a method called **main** which is the defined entry point for starting programs in Java, and can be used to create objects of other classes and call their methods.

Main

The main method

The main method has to be defined within a class in Java, and usually resides in a standalone class (here: Program)

```
public class Program
{
    public static void main(String[] args)
    {
        Student nick = new Student();
        Course computing = new Course();
        nick.enrol(computing);
    }
}
```

The main method is **static**

Dynamic members of a class would be accessed through an object (an instance). However, the main method is defined to be **static**, meaning that the method can be called through the class name (no object required).

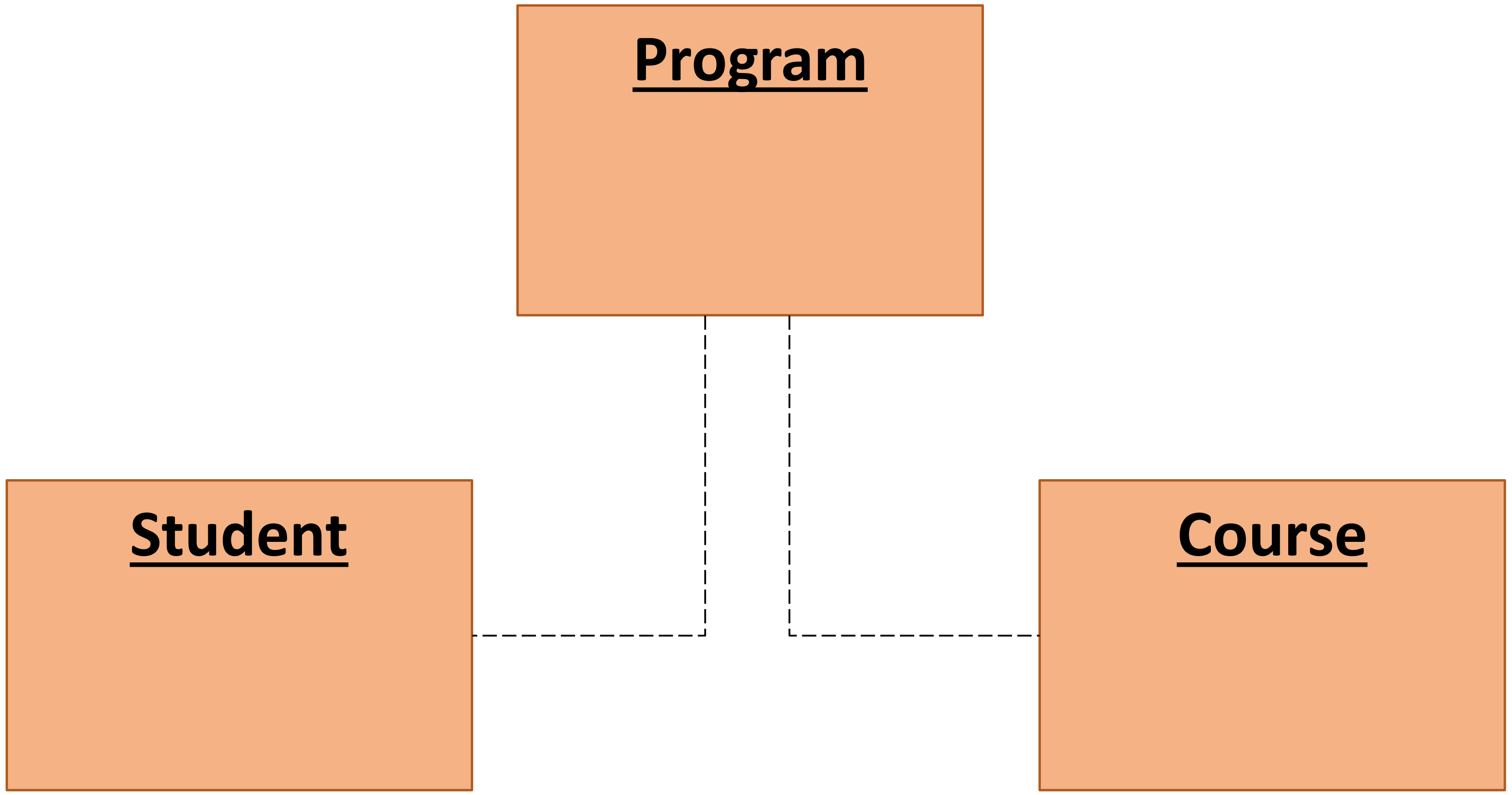
```
public class Program
{
    public static void main(String[] args)
    {
        …
    }
}
```

The main method

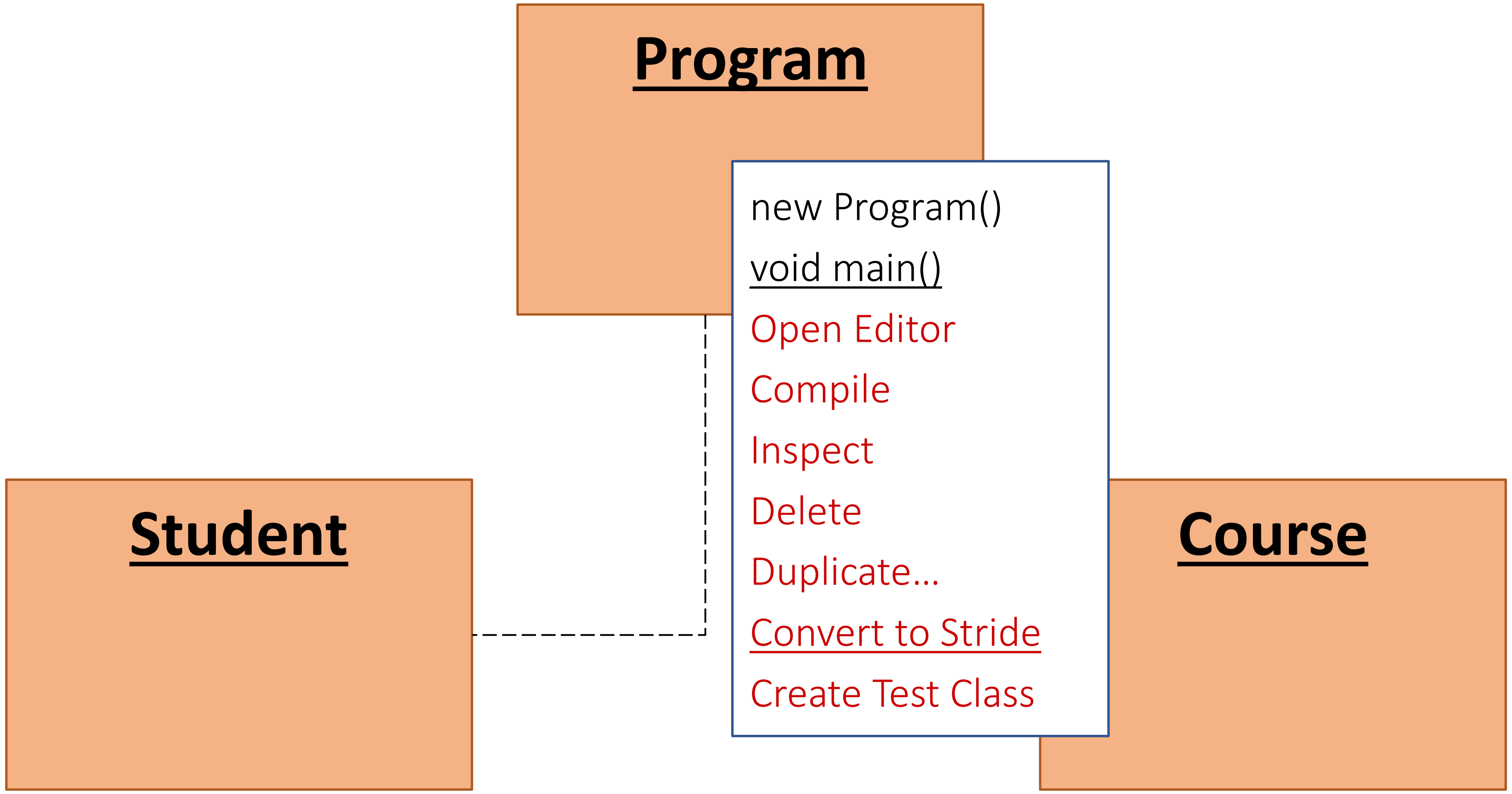
The main method accepts an argument (parameter) 'args'. The 'args' is an array, and optionally allows data to be passed in at compilation that can be used by main, but this is not required.

```
public class Program
{
    public static void main(String[] args)
    {
        …
    }
}
```

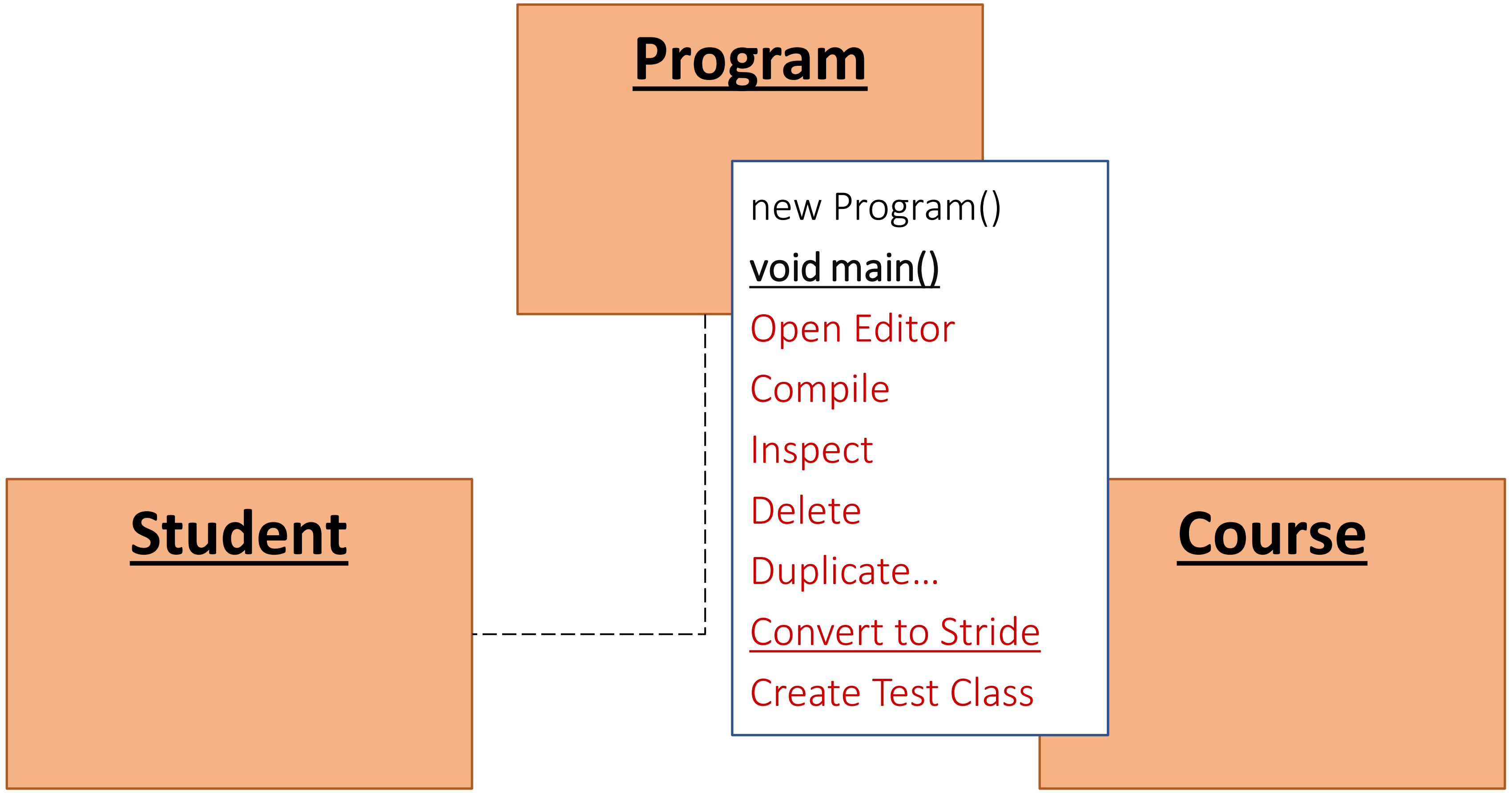
Class View



Class View – calling main



Class View – calling main



Outside of BlueJ

Coding a main method also allows Java files to be compiled and run outside of BlueJ.

The Java files that make up programs are compiled as an executable (application) so that they can run on any device, independent of an IDE.

User Interfaces

User interfaces

Up until now we’ve simulated the actions of a user by calling methods through objects on the bench, or hard-coded into the test methods.

In conjunction with allowing programs to be run independently of BlueJ, we also need to provide a means for users to interact with the program so it can be run by any user.

This would typically require an interface to enable users to enter information and choose options.

GUI and UI

We’re probably used to seeing Graphical User Interfaces (GUI) – either applications that we run on devices, or even web pages that provide a visually engaging and interactive ‘front-end’ to system.

It is possible to create a GUI for Java applications, however, in this module, we’ll focus on simulating a ‘front-end’ by creating a basic text-based menu of options to select from. This allows will allow for the user to select an option and interact with the program.

Example of a text-based UI

```
***************************
****** Stock Application ******
***************************
Select one of the following options:
1) Add a product
2) Print all products
3) Remove a product
4) Quit the program

Please enter your choice >
```

Input

Input via the Scanner

```
//1. import the Scanner class
import java.util.Scanner;
//2. create an object of the Scanner class
private Scanner reader = new Scanner(System.in);
//3. call methods on the object
 String input = reader.nextLine(); //returns a string
 int number = reader.nextInt(); //returns an int
```

Portion of the Scanner class

```
public class Scanner…
{
  /**
   * Advances this scanner past the current line and returns the input that was skipped.
   */
   public String nextLine()
   {
     …
   }
  /**
   * Scans the next token of the input as an int.
   */
   public int nextInt()
   {
      …
   }
}
```

Full documentation available at:
https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Input Validation

Validation

The process of validation checks to see where the user input adheres to **syntax rules** (correct data type) and the **logic** of the program (e.g. no negative values accepted, or input cannot be blank).

Remember that Java is a statically typed language and requires data to be stored in the type stated. It is not possible to store String data (indicated by a pair of speech marks: " ") in an integer variable. If attempted, this could cause an exception to be thrown or crash the program if not handled appropriately.

Building an InputReader Class

Therefore, as validation is essential, and input will be performed repetitively, we could create a class (InputReader) with methods that first output a prompt, then store the input, and also check that the input is valid (not empty, incorrect type etc).

```
import java.util.Scanner;
public class InputReader
{
    private Scanner reader;
    public InputReader()
    {
        reader = new Scanner(System.in);
    }
```

getString() method

```
public String getString(String prompt)
{
    String inputLine = null;
    boolean isValid = false;
    while(!isValid)
    {
      System.out.print(prompt);        // output prompt
      inputLine = reader.nextLine();   // wait for input
      if(!inputLine.isEmpty())
        isValid = true;
      else
        System.out.println("\nYour input is blank!\n");
    }
    return inputLine;
  }
```

getInt() method

```
public int getInt(String prompt)
 {
     int number = 0;
     boolean isValid = false;
     while(!isValid)
     {
        System.out.print(prompt);          // output prompt
        number = reader.nextInt();          // wait for input
        if(number >= 0)
           isValid = true;
        else
           System.out.println("\nYour input cannot be negative!\n");
     }
     return number;
  }
```

Input from the keyboard

```
***************************
****** Stock Application ******
***************************
Select one of the following options:
1) Add a product
2) Print all products
3) Remove a product
4) Quit the program

Please enter your choice >  add
```