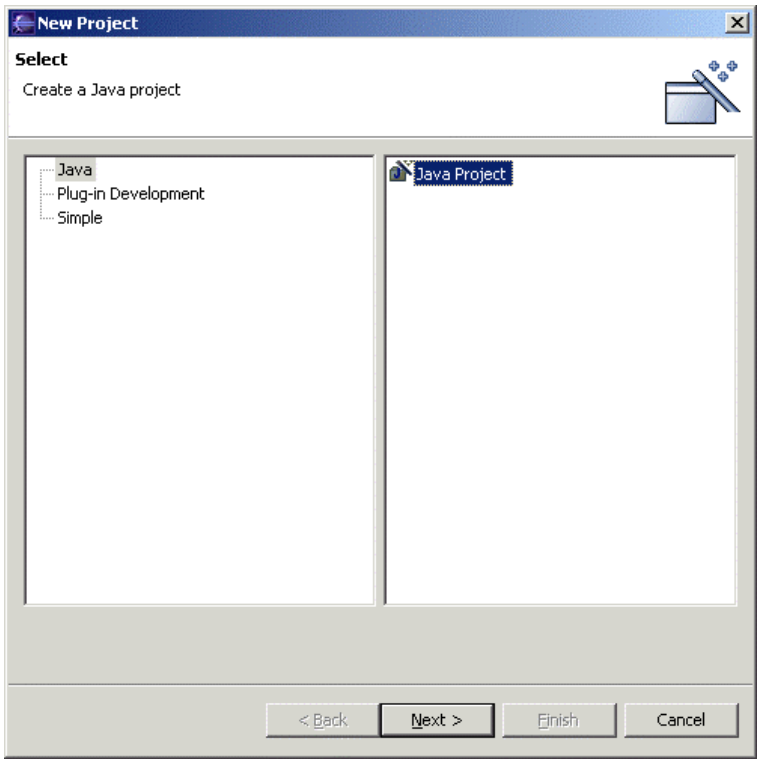
S8353: Java From the Very Beginning Part I - Exercises

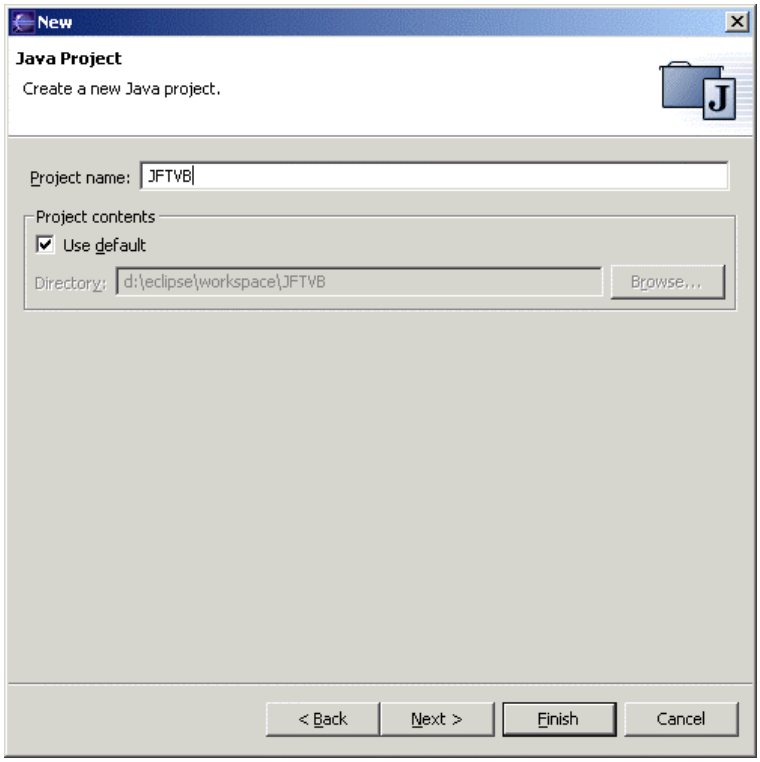
Ex. 1 Hello World

This lab uses the Eclipse development environment which provides all of the tools necessary to build, compile and run Java applications. The lab instructor will demonstrate how to load Eclipse on the lab machines.

To import the exercises, once Eclipse has loaded, create a new project by selecting **File** -> **New** -> **Project**. A wizard will appear, as shown below. Ensure **Java Project** is selected then click **Next**.

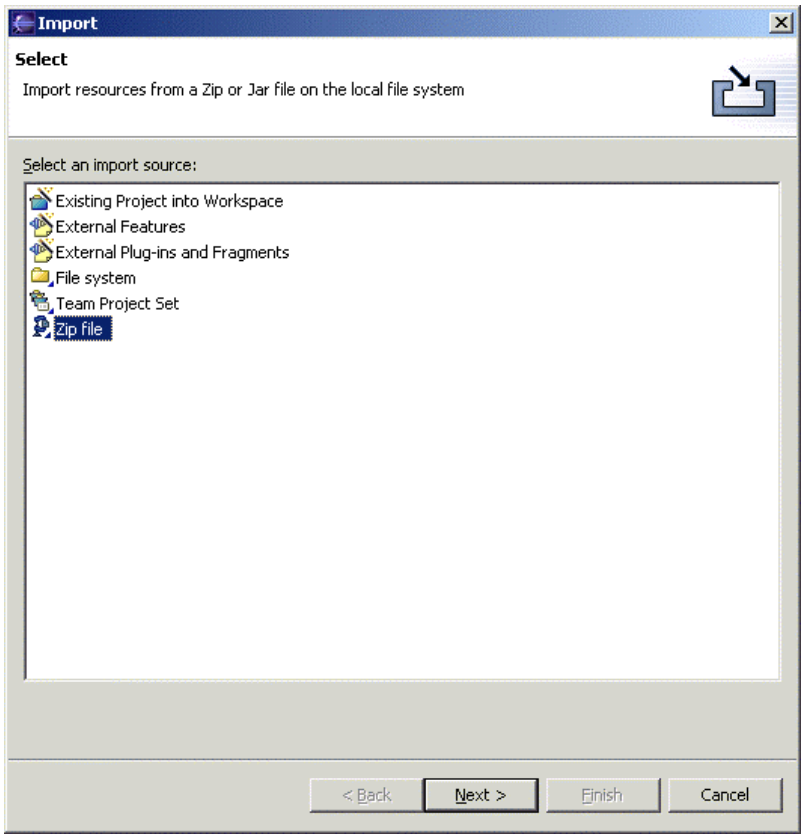


In the **Project Name** text box, enter **JFTVB**, which stands for "Java From the Very Beginning" and click **Finish**. This will create a new project in Eclipse, which will contain our work.



If you are asked if you would like to open the Java perspective, accept this.

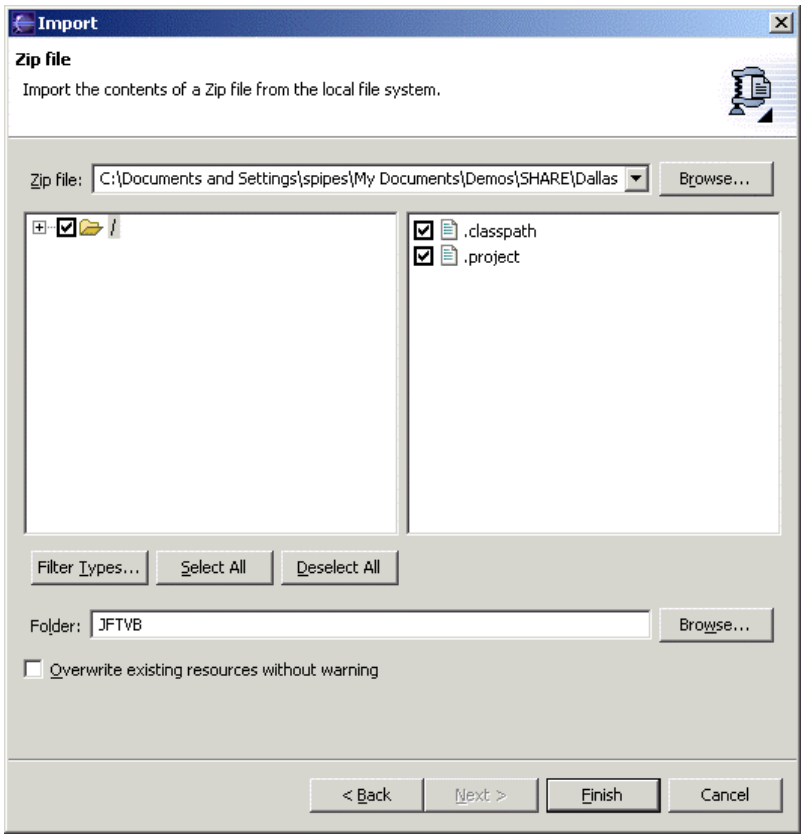
Once the project has been created, select **File -> Import** from the menu bar. An import wizard will appear.



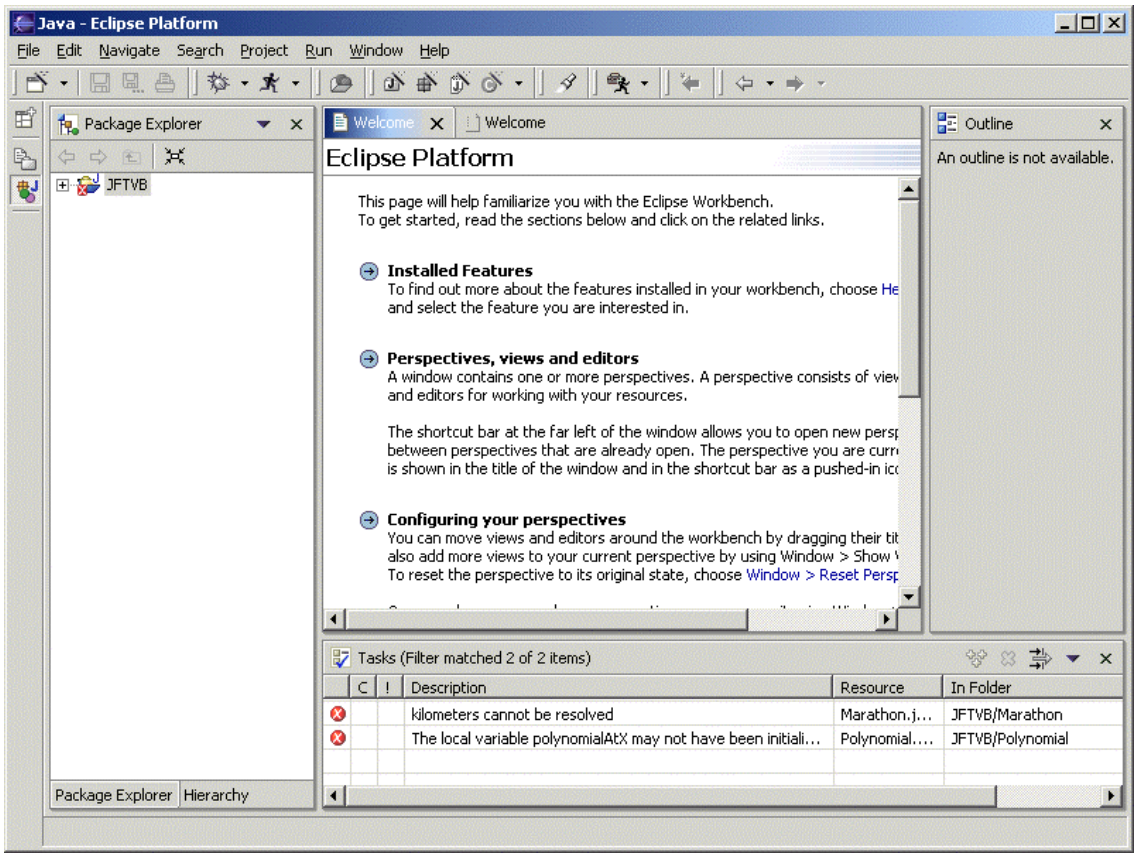
Select **File System** and click **Next**.
Click the **Browse** button to the right of this text box and search for the **JFTB** folder under **Desktop > Javalabs > Projects.**

If the file loaded successfully, entries will appear in the windows below. These show which parts of the exercise will be imported into Eclipse. Ensure that the top-level folder (which is shown with the / character) is ticked, as shown in the diagram below.
Enter **JFTVB** in the **Folder** text box and click **Finish** to import the project.
If you receive any messages asking about whether Eclipse should overwrite certain files, such as the .classpath file, then click **Yes To All**.

Note: some of the screen shots say **Zip File** instead of **File System**… Other that that they should look like what you see. For all the labs, the exercises will be found in the **File System: Desktop > Javalabs > Projects folder.**



Once the project has been imported, you will be presented with the Java perpective, as shown in the diagram below, which shows the project you have just imported. This project can be expanded, by clicking the “plus” symbol to the left of the project name.
Also notice the tasks view in the bottom-right of the screen. This reports that two compilation errors currently exist in the project. These will be resolved as part of the exercises.



If you are not familiar with Eclipse, it is recommended that you use the **Java perspective**. Ask the instructor if you are not sure how to enable this perspective.

1) Open the **HelloWorldApp.java** file which is in the **HelloWorldApp** folder.

2) The code below should appear (there is an extra line at the top of the code that defines a package for this code). Complete the program by adding the code required to say "Hello World!" (as displayed on the overheads and in your notes).

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
  // Add the body of your class here
  // ...

}
```

3) Save your changes by selecting **File -> Save HelloWorldApp.java** or by using the key shortcut **CTRL+S**. Eclipse will compile the application automatically and will present any compiler errors in the **Tasks** window in the bottom-right corner of the window.

4) The program should now be run. Ensuring that the **HelloWorldApp.java** file is highlighted in the **Package Explorer** view, click the arrow on the "Run" icon  and

select **Run As -> 2. Java Application**. This will start the Java runtime and run the HelloWorldApp application.
The output from this will be displayed in the **Console** view in the bottom right-hand window.

If you did not receive any error messages and the message “Hello World!” appeared in the console view then you have successfully completed the first part of this exercise. You are now able to work through the rest of this lab using the Eclipse environment.

Ex. 2 Marathon

1) Open the **Marathon.java** file which is in the **Marathon** folder.

2) You now need to declare some variables to use later on in the program. Find the place in the source code where it says:

```
// TO DO:
    // Declare two integer variables, miles and yards, and one double
variable,
    // kilometres
```

and immediately underneath insert appropriate declarations for the three variables.

3) The next step is to initialise the miles and yards variables to hold the number of miles and yards in a marathon respectively. Find the place in the source code where it says:

```
// TO DO:
    // Set miles to 26, and yards to 385
```

and add some code to set these two variables to the appropriate values.

4) Now we are ready to do the calculation from miles and yards to kilometres. Find the place in the code where it says:

```
// TO DO:
    // Write an expression to calculate kilometers from miles and yards.
    // Save the result of the expression in the variable kilometers.
    // One mile is 1.609 kilometers
    // There are 1760.0 yards in a mile
```

and insert some code to calculate the number of kilometres in a marathon. When dealing with the yards, do you need integer or floating point division? How does the compiler know which to use?

5) Save your code using the menu or keyboard shortcut as described in the previous exercise.

7) Run the Marathon program and take a note of the answer (which is displayed in the console view).

8 (optional)) If you have time, look at the sample solutions provided for the Marathon exercise. Three versions are provided. The first program is the solution to the exercise you have just completed. The second program shows how to improve the code by using constants for the fixed parameters. What is the Java keyword that introduces a constant?

The third program goes a stage further and formats the answer to three decimal places. You will not understand all of this program at this stage, but it introduces you to a small part of Java's rich class library and shows how you can import and make use of classes within it.

Java from the very beginning Part I- Sample Solutions

Ex. 1 Hello World

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Ex. 2 Marathon

```
/**
 * Marathon - a simple class to calculate the distance of a marathon in
kilometers
 */
class Marathon {

  /**
   * Main method called when we invoke the interpreter
   */
  public static void main(String[] args) {
    // TO DO:
    // Declare two integer variables, miles and yards, and one double
variable,
    // kilometers (be careful with the spelling)
    int miles;
    int yards;
    double kilometers;

    // TO DO:
    // Set miles to 26, and yards to 385
    miles = 26;
    yards = 385;

    // TO DO:
    // Write an expression to calculate kilometers from miles and yards.
    // Save the result of the expression in the variable kilometers.
    // One mile is 1.609 kilometers
    // There are 1760.0 yards in a mile
    kilometers = 1.609 * (miles + (yards / 1760.0));

    // Print the answer
    System.out.println("A marathon is " + kilometers + " kilometers.");

  }  // end of main method

} // end of marathon class
```

Improved version using constants:

```
/**
 * Marathon2 - a simple class to calculate the distance of a marathon in
kilometers.
 * Uses constants for fixed values.
 */
class Marathon2 {
```

```
  /**
   * Main method called when we invoke the interpreter
   */
  public static void main(String[] args) {
    // Declare constants to use in the calculation:
    // final variables cannot be changed once they have been initialised
    final int miles = 26;
    final int yards = 385;
    final double kilometersPerMile = 1.609;
    final double yardsPerMile = 1760.0;

    double kilometers;

    // TO DO:
    // Write an expression to calculate kilometers from miles and yards.
    // Save the result of the expression in the variable kilometers.
    // One mile is 1.609 kilometers
    // There are 1760.0 yards in a mile
    kilometers = kilometersPerMile * (miles + (yards / yardsPerMile));

    // Print the answer
    System.out.println("A marathon is " + kilometers + " kilometers.");

  }  // end of main method

} // end of marathon class
```

Improved version with control of output format

```
/**
 * Marathon3 - a simple class to calculate the distance of a marathon in
kilometers
 * Uses defined constants, and prints the output to three decimal places.
 */

// import the NumberFormat class into our program since we want to control
the
// appearance of the formatted output
import java.text.NumberFormat;

class Marathon3 {

  /**
   * Main method called when we invoke the interpreter
   */
  public static void main(String[] args) {
    // Declare constants to use in the calculation:
    // final variables cannot be changed once they have been initialised
    final int miles = 26;
    final int yards = 385;
    final double kilometersPerMile = 1.609;
    final double yardsPerMile = 1760.0;

    double kilometers;

    // TO DO:
    // Write an expression to calculate kilometers from miles and yards.
    // Save the result of the expression in the variable kilometers.
    // One mile is 1.609 kilometers
    // There are 1760.0 yards in a mile
    kilometers = kilometersPerMile * (miles + (yards / yardsPerMile));

    // Print the answer, to three decimal places
    // Get a number formatter
    NumberFormat numberFormatter = NumberFormat.getNumberInstance();
    // Tell it to use three decimal places
    numberFormatter.setMaximumFractionDigits(3);
```

```
    // Format the number
    String strKilometers = numberFormatter.format(kilometers);
    // Print the answer
    System.out.println("A marathon is " + strKilometers + " kilometers.");

  }  // end of main method

} // end of marathon class
```