School of Computing Sciences

CMP-4008Y — Programming 1

LAB EXERCISES 1 — Getting Started

In this laboratory exercise, we will investigate the use of IntelliJ IDEA to create a simple Java application that prints a recipe for chocolate chip cookies on the screen:

```
Recipe : Chocolate chip cookies (makes 12 cookies)

Ingredients: 4oz sugar
             4oz butter
             8oz self raising flour
             1   egg
             vanilla essence
             4oz chocolate chips

step 1 : Mix sugar, butter and self raising flour thoroughly
step 2 : Stir in egg, a few drops of vanilla essence and chocolate chips.
step 3 : Roll out mixture and cut out cookies.
step 4 : Bake in hot oven (Gas Mark 6) for 10-12 minutes.
```

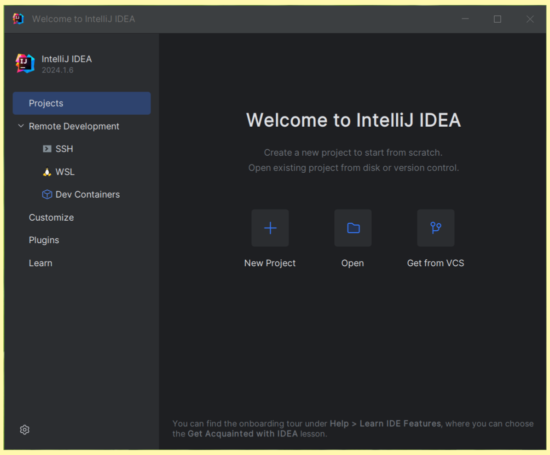
In order to do this you will need to:

1. Create a `.java` file that contains the Java code which will produce this output.
2. Compile the code.
3. Run the application to test the output.

While you can write the source code (the `.java` file) using any text editor and compile and run the program using command line tools, you may find it easier to use an Integrated Development Environment (IDE) such as IntelliJ, which not only allows you to edit the code, but also to compile and run the application. It has an easy to use interface, with syntax highlighting, code completion, code templates and editor hints to help avoid mistakes.

IntelliJ IDEA is a cross-platform IDE that has two editions: "Ultimate" (commercial licence) for web and enterprise development and "Community" (Open-source, Apache 2.0 licence) for JVM and Android development. On CMP laboratory machines, it can be launched using the Start menu:

Start → CMP Development Tools
→ IntelliJ



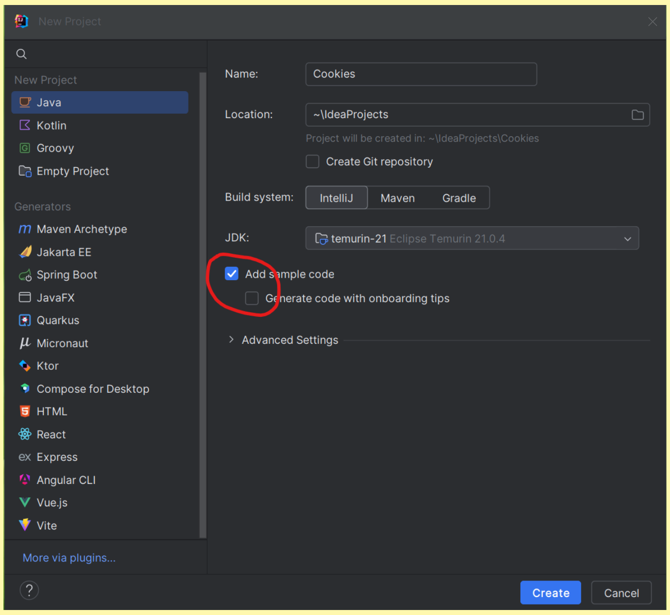
Welcome to IntelliJ IDEA

Create a new project to start from scratch.
Open existing project from disk or version control.

You can find the onboarding tour under **Help > Learn IDE Features**, where you can choose the **Get Acquainted with IDEA** lesson.

Figure 1: Starting up IntelliJ Idea

This will open the window shown in Figure 1. (The screenshots in this document were all obtained from a slightly different version of IntelliJ running on Windows, so there may be slight variations in appearance on other operating systems.) Initially there is no project. You have a choice of: creating a new project, importing a project (if you have already created a project using another IDE), opening an existing project or checking out a project from a version control repository.



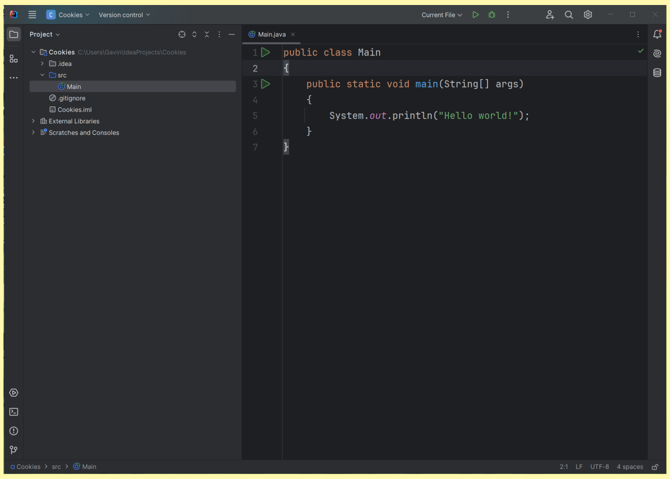
Name: Cookies
Location: ~\IdeaProjects
Project will be created in: ~\IdeaProjects\Cookies
[ ] Create Git repository
Build system: [✓ IntelliJ] Maven Gradle
JDK: temurin-21 Eclipse Temurin 21.0.4
[✓] Add sample code
[ ] Generate code with onboarding tips
Advanced Settings

Figure 2: Creating a New Project

In order to create a new Java application, you must first create a new project. This can be done by clicking on the “Create New Project” button. You should now see the window shown in Figure 2. In the left hand panel, select the language or system for the project. In this case, Java is selected. You may want to deselect “Generate code with onboarding tips” as that will generate code with comments giving hints about how to use the IDE. Select “Add sample code” though as then the IDE will generate a project with some simple working code as a useful starting point.

This exercise is about creating a Java application that will print out the recipe for chocolate chip cookies, so you might want to name the project “Cookie”. For the project location, you need to specify a folder in which to save your work. You can either type this in directly in the Location text field, or you can click on the folder icon, to open a file browser and navigate your way to the required folder. (The folder should be empty or your files may be overwritten!) You can use the file browser interface to create a new folder, if required. Then click “Create” to complete the new project setup.

Once you have successfully created your new project, you should now see something like Figure 3. This project has one `.java` file, called `Main.java`, containing a single `public class`, called `Main`. This is loaded into the editor so we can start working straight away. The left hand panel allows you to navigate around your project files, so that you can load other files into the editor. The code implements the familiar “Hello, world!” program that we saw in lecture 2. The indentation may be somewhat different on your computer - try editing it so that it is formatted as shown in the listing below:

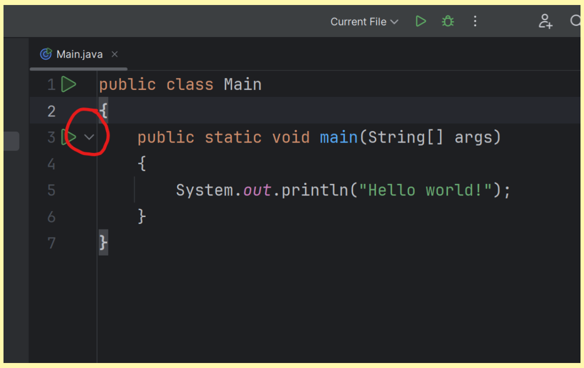


```
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Hello world!");
    }
}
```

Figure 3: The new project is ready to be edited

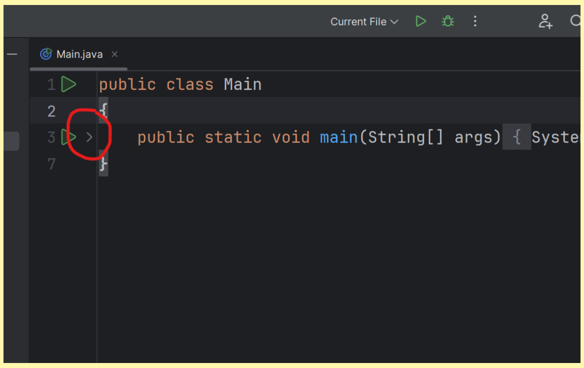
```java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println(Hello world!");
    }
}
```

Note that it is possible to collapse parts of the code, such as the comment or function blocks, to hide them whilst you are editing other parts of the file. To collapse these blocks, click on the down triangle box in the margin, Figure 4(a), and click on the right arrow in the margin, Figure 4(b), to make them reappear again. Note, the arrows may only appear when the pointer is hovering over the margin.



```java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Hello world!");
    }
}
```

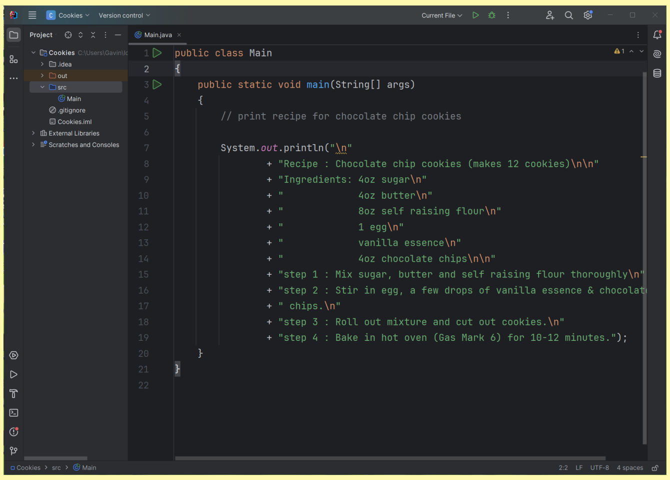
(a)



```java
public class Main
{
    public static void main(String[] args) { Syste
}
```

(b)

Figure 4: Collapsing Code Blocks

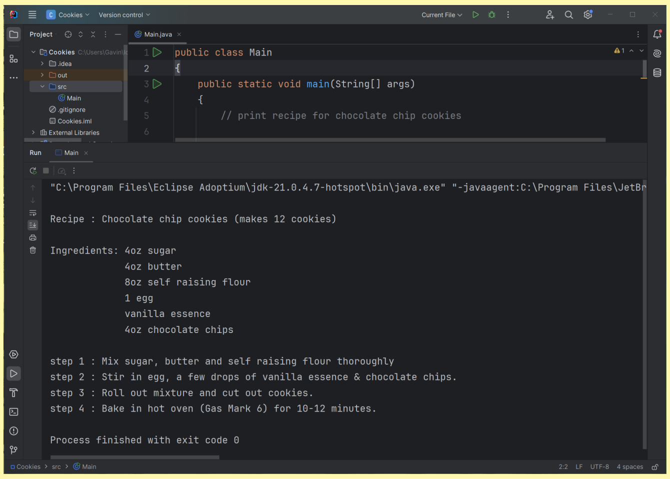


```java
public class Main
{
    public static void main(String[] args)
    {
        // print recipe for chocolate chip cookies

        System.out.println("\n"
                + "Recipe : Chocolate chip cookies (makes 12 cookies)\n\n"
                + "Ingredients: 4oz sugar\n"
                + "             4oz butter\n"
                + "             8oz self raising flour\n"
                + "             1 egg\n"
                + "             vanilla essence\n"
                + "             4oz chocolate chips\n\n"
                + "step 1 : Mix sugar, butter and self raising flour thoroughly\n"
                + "step 2 : Stir in egg, a few drops of vanilla essence & chocolat
                + " chips.\n"
                + "step 3 : Roll out mixture and cut out cookies.\n"
                + "step 4 : Bake in hot oven (Gas Mark 6) for 10-12 minutes.");
    }
}
```

Figure 5: Editing Code

The `main` method initially just prints "Hello world!" on the console. This should be changed to something a bit more meaningful, for example: `print recipe for chocolate chip cookies`. You now need to enter the code that will produce the chocolate chip cookie recipe. This just involves a `System.out.println` statement, as shown in Figure 5 on the right.

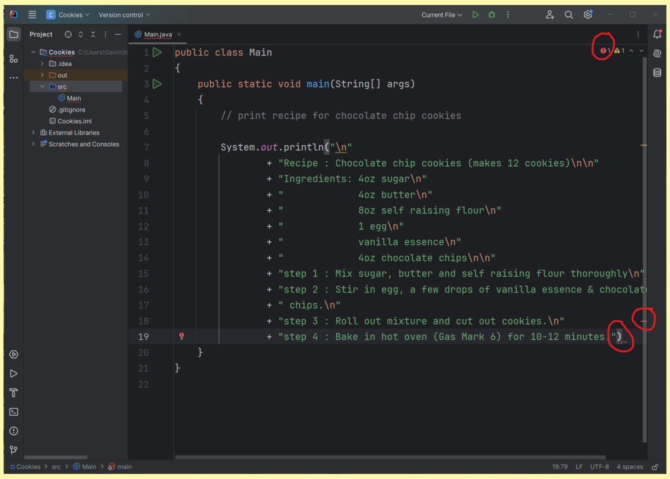
Once you have finished editing your code, you can click on the "Run" button ▷ (or use the Run menu) to build and run the application. This opens a new panel (the output panel) at the bottom of the window showing the text that the application wrote to `STDOUT` (via `System.out.println`). The final line reads "Process finished with exit code 0" which indicates that the application ran successfully. If your application returns a exit code 0, then you have successfully completed the first part of this exercise.



```
public class Main
{
    public static void main(String[] args)
    {
        // print recipe for chocolate chip cookies
```

```
"C:\Program Files\Eclipse Adoptium\jdk-21.0.4.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBr

Recipe : Chocolate chip cookies (makes 12 cookies)

Ingredients: 4oz sugar
             4oz butter
             8oz self raising flour
             1 egg
             vanilla essence
             4oz chocolate chips

step 1 : Mix sugar, butter and self raising flour thoroughly
step 2 : Stir in egg, a few drops of vanilla essence & chocolate chips.
step 3 : Roll out mixture and cut out cookies.
step 4 : Bake in hot oven (Gas Mark 6) for 10-12 minutes.

Process finished with exit code 0
```

Figure 6: Successful compilation

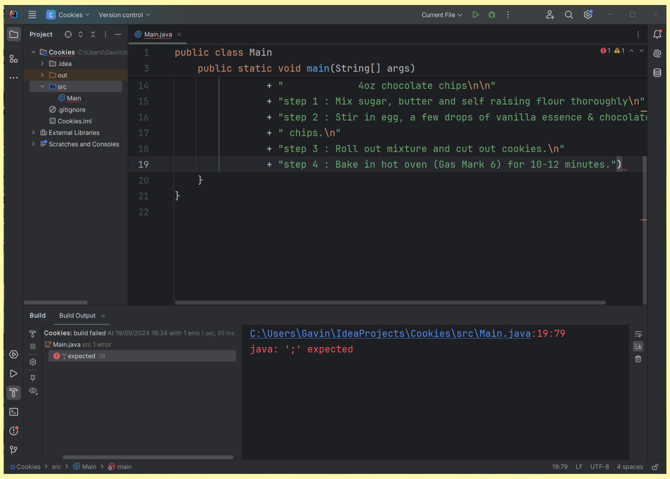


```
1  public class Main
2  {
3      public static void main(String[] args)
4      {
5          // print recipe for chocolate chip cookies
6
7          System.out.println("\n"
8                  + "Recipe : Chocolate chip cookies (makes 12 cookies)\n\n"
9                  + "Ingredients: 4oz sugar\n"
10                 + "             4oz butter\n"
11                 + "             8oz self raising flour\n"
12                 + "             1 egg\n"
13                 + "             vanilla essence\n"
14                 + "             4oz chocolate chips\n\n"
15                 + "step 1 : Mix sugar, butter and self raising flour thoroughly\n"
16                 + "step 2 : Stir in egg, a few drops of vanilla essence & chocolat
17                 + " chips.\n"
18                 + "step 3 : Roll out mixture and cut out cookies.\n"
19                 + "step 4 : Bake in hot oven (Gas Mark 6) for 10-12 minutes.")
20     }
21 }
22
```

Figure 7: The semi-colon has been omitted at the end of the `System.out.println` command.

Now suppose you have made an error, say you have missed out the semi-colon at the end of the `System.out.println` command. This is illustrated in Figure 7. Notice that the line of code where the semi-colon is missing now has a wavy line beneath it. This is a useful indicator to let you know you have made a mistake. There are also a couple of other indicators which you may have noticed: there is a small horizontal bar marking the line where the error occurs, an exclamation mark in a red circle above in the right hand corner.

Suppose you haven't noticed the error, and try to compile and run your program. In this case the compilation will fail. The status shown in the bottom left panel of the screen reads "Build failed ... with 1 error ...". The message "Error:(19,79) java: ';' expected" is shown in the bottom right panel. This indicates that a problem occurred on line 19, column 79 and the cause of the error is a missing semi-colon. If you double-click on the error message, the cursor will be moved to location in the source code.



```
1  public class Main
3      public static void main(String[] args)
14                 + "             4oz chocolate chips\n\n"
15                 + "step 1 : Mix sugar, butter and self raising flour thoroughly\n"
16                 + "step 2 : Stir in egg, a few drops of vanilla essence & chocolat
17                 + " chips.\n"
18                 + "step 3 : Roll out mixture and cut out cookies.\n"
19                 + "step 4 : Bake in hot oven (Gas Mark 6) for 10-12 minutes.")
20     }
21 }
22
```

```
Cookies: build failed At 19/09/2024 16:34 with 1 erro 1 sec, 85 ms
  Main.java src 1 error
    ';' expected :19
```

```
C:\Users\Gavin\IdeaProjects\Cookies\src\Main.java:19:79
java: ';' expected
```

Figure 8: Build Failed

Once you have fixed the problem, rebuild your project. Don't worry if you get compilation errors - cycles of compiling and fixing errors is a completely normal part of programming and the error messages are *intended* to help you identify the problem. Experiment with the buttons on the toolbar, and the menus, which implement frequently performed actions, such as building the current project. Familiarity with these buttons will improve your productivity as a programmer.

The second part of this exercise requires you to modify the code so that the application requests the number of batches of 12 cookies from the user, and modifies the output accordingly. The `Scanner` class provides methods to read user input. Study the programs covered in the lecture for examples of reading from the keyboard using `Scanner`.

Gavin Cawley
20th September 2024