

Creating and Using JAR Files

A very common way of making your applications available to clients is to convert them to JAR files. A JAR file (short for Java Archive) has the extension `.jar` and is simply a compressed file.

It is possible to create JAR files from the command prompt with the **jar.exe** tool. However, this can be error-prone and is difficult for beginners. It is far easier to create JAR files by using an IDE, which we show you below.

There are two main ways in which jar files can be used, and we deal with these in turn.

1. Non-executable JAR files

When an application is being compiled, one or more classes may require the presence of other classes, as we have seen throughout the text. These classes will, of course, all need to be available when the application is compiled. It is possible to provide some or all of these classes in the form of a JAR file.

As an example we will use some of the classes we created in chapters 7 and 8. Here we developed a class called `RectangleTester`:

RectangleTester.java

```
public class RectangleTester
{
    public static void main(String[] args)
    {
        double RectangleLength, RectangleHeight;

        Rectangle myRectangle;

        System.out.print("Please enter the length of your Rectangle: ");
        RectangleLength = EasyScanner.nextDouble();
        System.out.print("Please enter the height of your Rectangle: ");
        RectangleHeight = EasyScanner.nextDouble();

        myRectangle = new Rectangle(RectangleLength, RectangleHeight);

        System.out.println("Rectangle length is " + myRectangle.getLength());
        System.out.println("Rectangle height is " + myRectangle.getHeight());
        System.out.println("Rectangle area is " + myRectangle.calculateArea());
        System.out.println("Rectangle perimeter is " + myRectangle.calculatePerimeter());
    }
}
```

As can be seen, this class requires two classes to be on the classpath during the compilation - `Rectangle` and `EasyScanner`. We will create a JAR file containing these two classes, and then show how this can be incorporated into a project when we compile and run `RectangleTester`.

We will use IntelliJ® as our example IDE - other IDEs will have similar procedures, and you can find these from the documentation for the particular IDE.

In Figure 1 we have created an IntelliJ project called `rect` containing the two files `Rectangle` and `EasyScanner`.

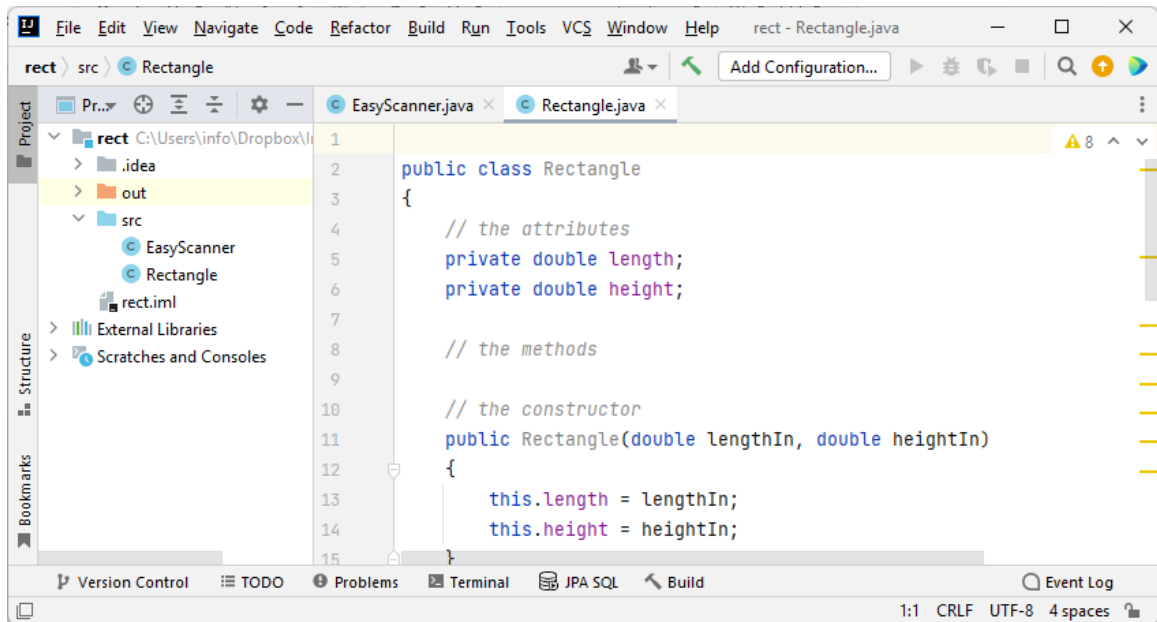


Figure 1

To create a JAR file containing these two classes select **File > Project Structure > Artifacts** then click on the **+** sign and choose **JAR > From modules with dependencies**. See Figure 2

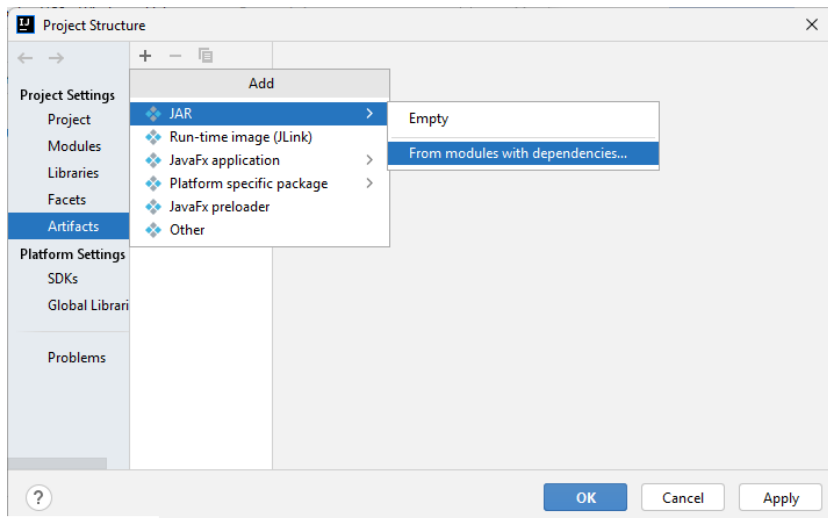


Figure 2

You will now see the screen shown in Figure 3.

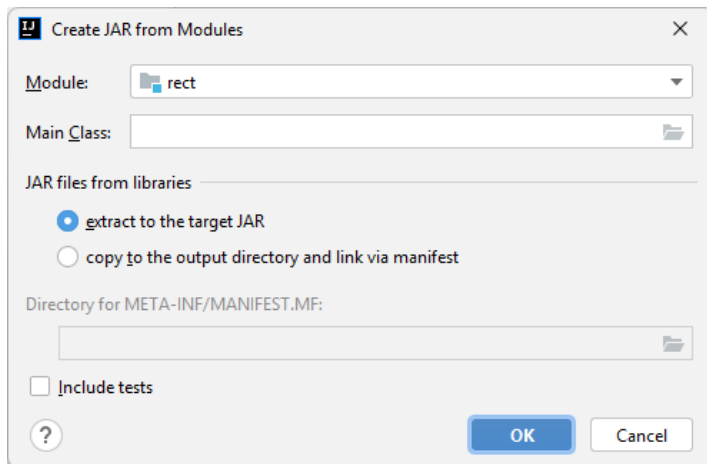


Figure 3

You do not need a main class because we are not creating an executable file here, so simply press **OK** and proceed.

In order to build the JAR file, select **Build > Build Artifacts**. Then select **Build** from the **Action** menu (Figure 4).

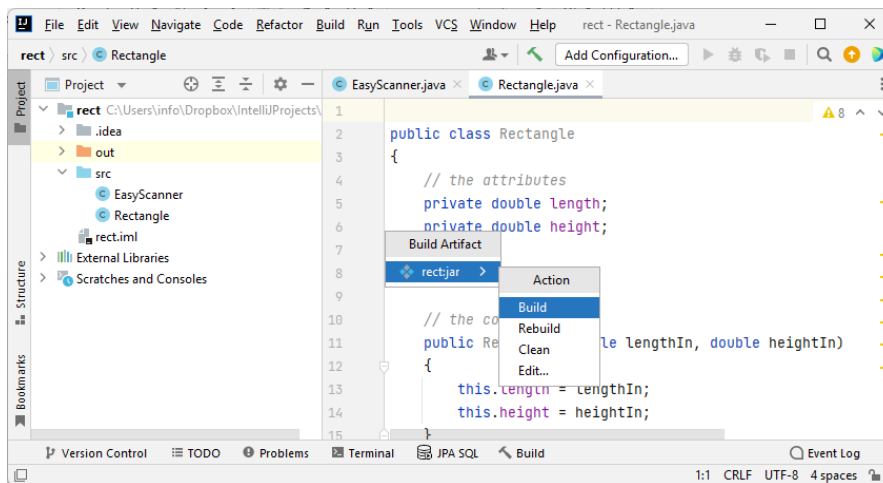


Figure 4

As shown in Figure 5, the JAR file, `rect.jar` will be located in the **out > artifacts** directory:

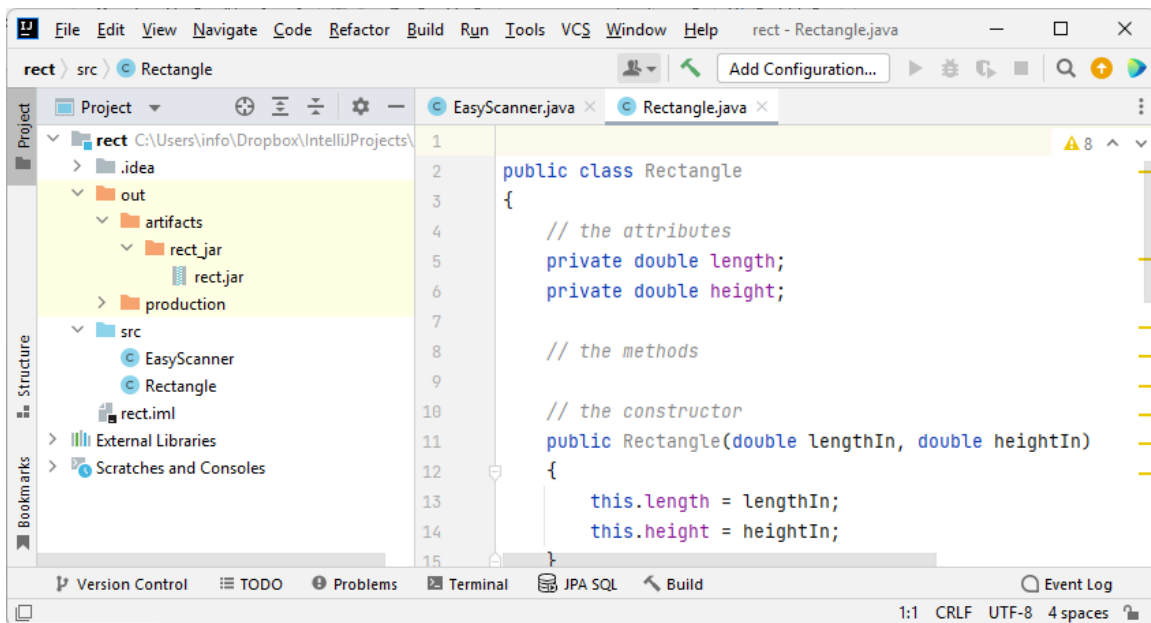


Figure 5

We are now in a position to use this JAR file. We have created a project called `JarTester` in IntelliJ and added the `RectangleTester.java` file to the project, as shown in Figure 6.

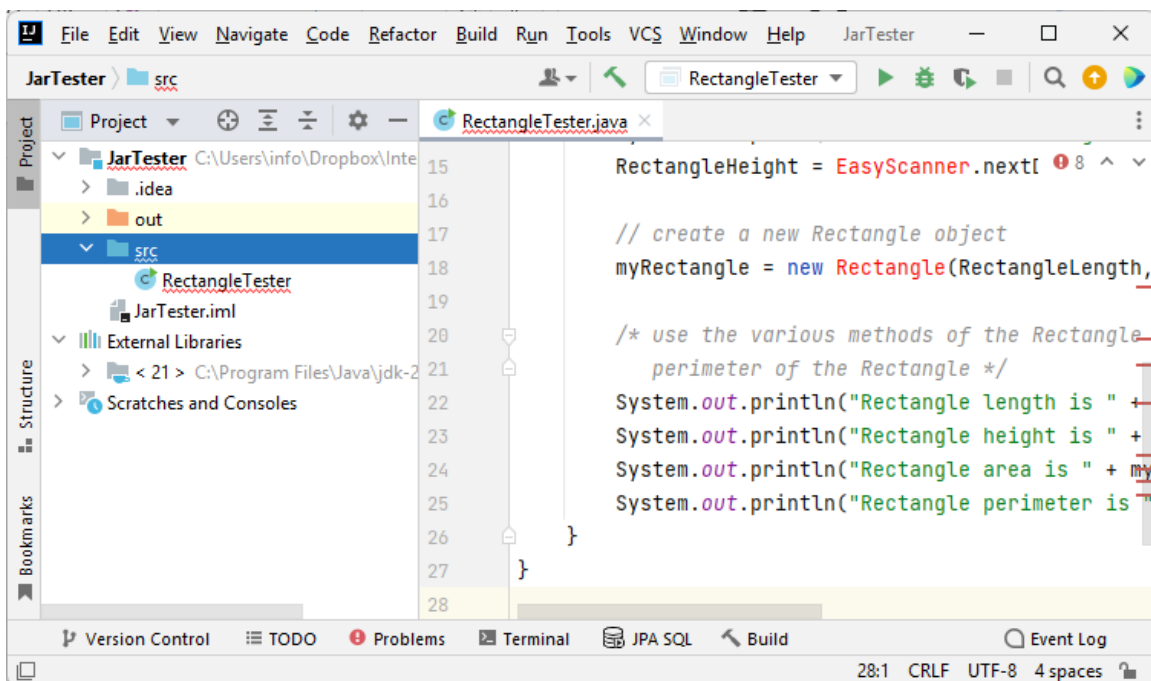


Figure 6

As you can see from Figure 6 there are compiler errors because the files on which this class depends are not present on the classpath. But we can add our `rect.jar` file to the dependencies as follows:

Select **File > Project Structure**, highlight **Libraries** then press the **+** sign. Choose **java** and you will then be able to select the JAR file you wish to add (`rect.jar` in this case). See Figure 7.

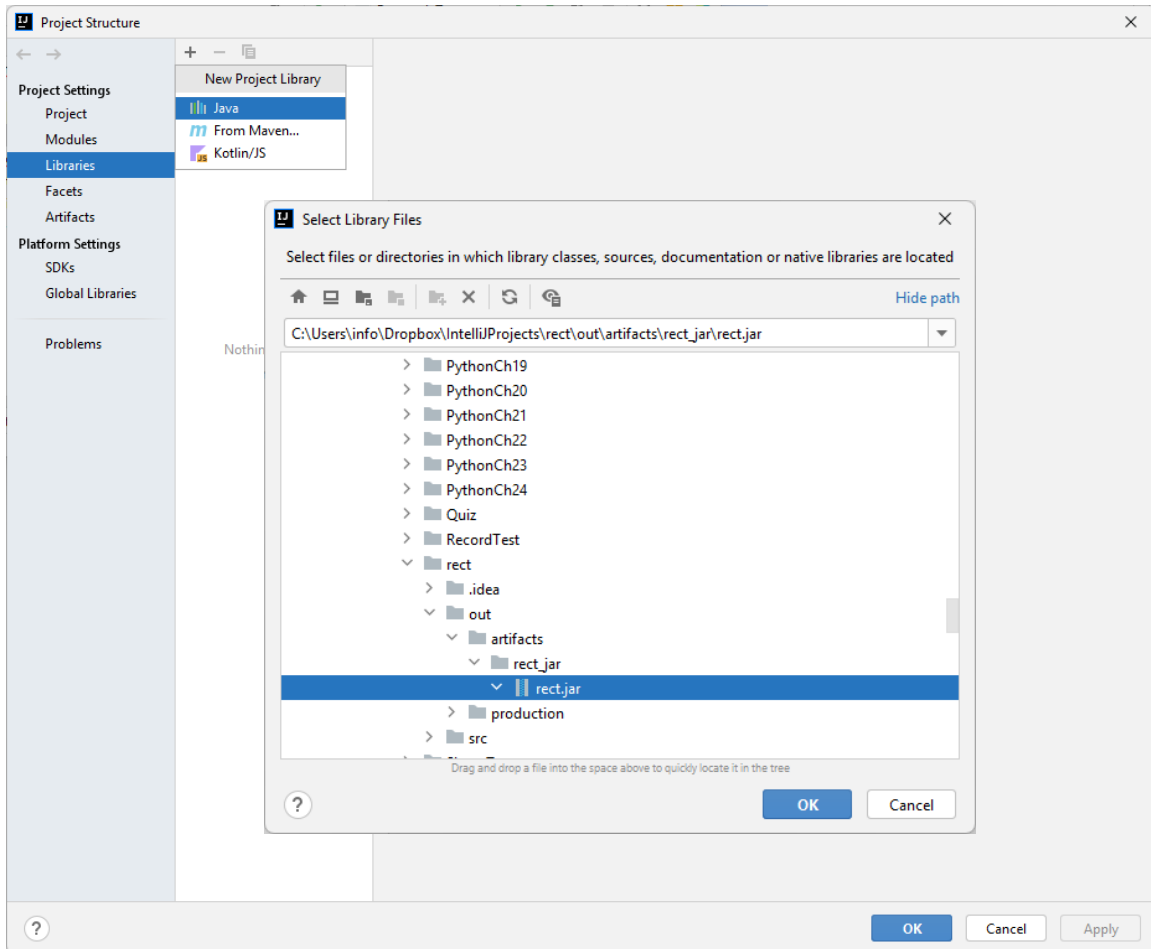


Figure 7

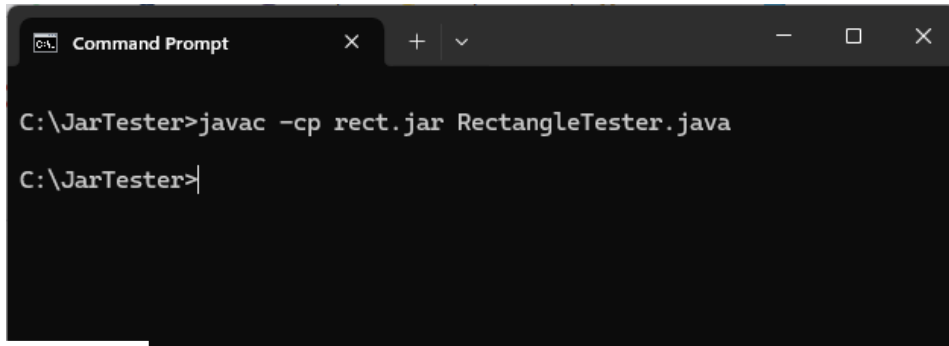
Once this has been added you will be able to compile and run `RectangleTester` successfully.

You have seen a practical example of this in action in section 23.7 of chapter 23 when we added a JDBC driver, in the form of a JAR file, to our project configuration.

Our rect.jar file could also be used to compile and run RectangleTester from the command line. We will assume that rect.jar and RectangleTester.java are both located in a folder called JarTester. Working from within this directory, we could compile RectangleTester with the following command:

```
javac -cp rect.jar RectangleTester.java
```

This is illustrated in Figure 8.

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt" and "C:\\". The command prompt shows the directory "C:\JarTester" and the command "javac -cp rect.jar RectangleTester.java" being executed. The prompt is now "C:\JarTester>".

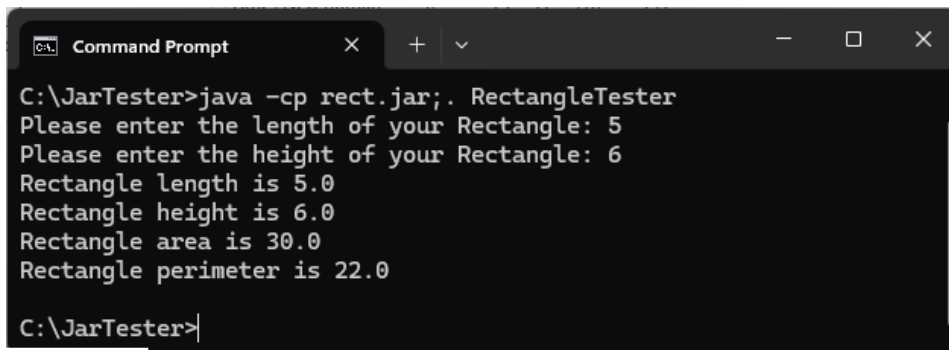
```
Command Prompt
C:\JarTester>javac -cp rect.jar RectangleTester.java
C:\JarTester>
```

Figure 8

We had to use the cp (classpath) argument to indicate that the required files are contained in the archive file, rect.jar

This will create the RectangleTester.class file, which can be run with the following line as shown in Figure 9.

```
java -cp rect.jar;. RectangleTester
```

A screenshot of a Windows Command Prompt window. The title bar says "Command Prompt" and "C:\\". The command prompt shows the directory "C:\JarTester" and the command "java -cp rect.jar;. RectangleTester" being executed. The output shows prompts for length and height, and then displays the calculated area and perimeter. The prompt is now "C:\JarTester>".

```
Command Prompt
C:\JarTester>java -cp rect.jar;. RectangleTester
Please enter the length of your Rectangle: 5
Please enter the height of your Rectangle: 6
Rectangle length is 5.0
Rectangle height is 6.0
Rectangle area is 30.0
Rectangle perimeter is 22.0
C:\JarTester>
```

Figure 9

Note that the current directory (.) must be included in the classpath if the main class is outside the JAR. The classpaths are separated with a semi-colon.

2. Executable JAR files

A JAR file can be made to be executable so that it can be run from the command line. Figure 10 shows a project that has been created in IntelliJ. It contains four classes that were discussed in chapters 7 and 8, and which make up a bank application. The main class is `BankApplication`.

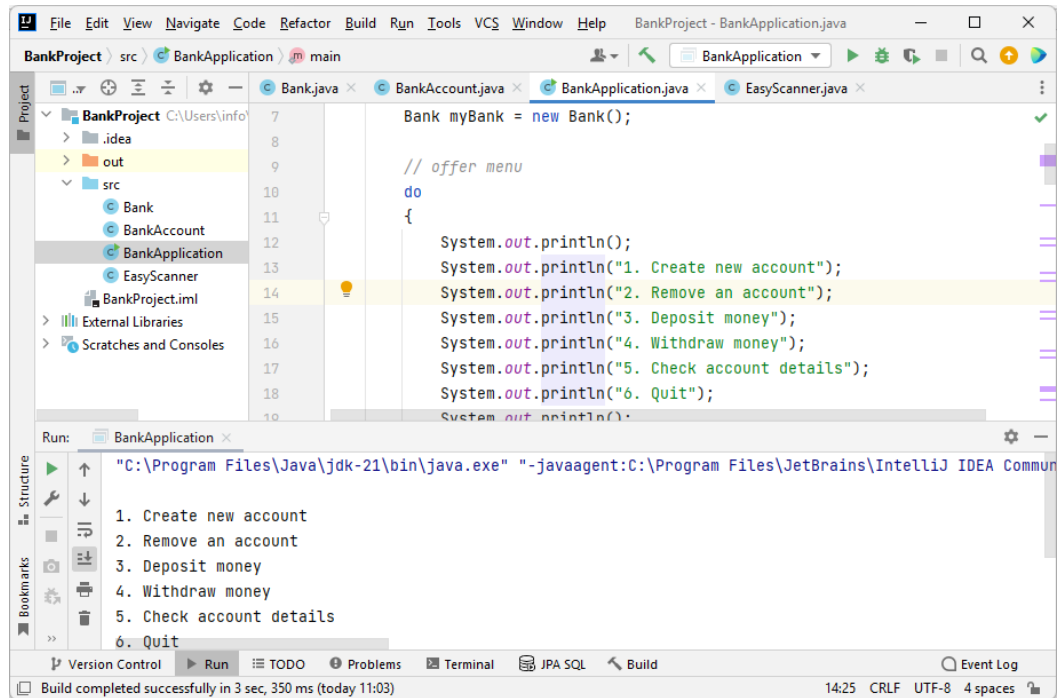


Figure 10

To create an executable JAR file we proceed as before (that is: select **File > Project Structure > Artifacts**, click **+**, and choose **JAR > From modules with dependencies**), but this time, when we create our JAR file we need to name the main class, as shown in Figure 11.

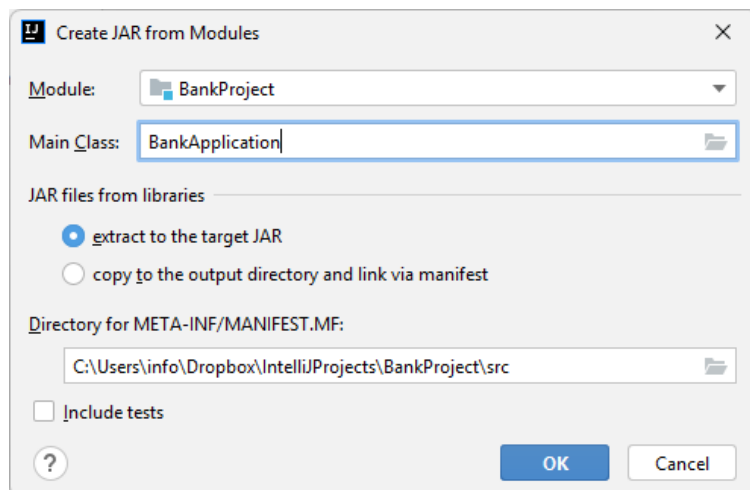


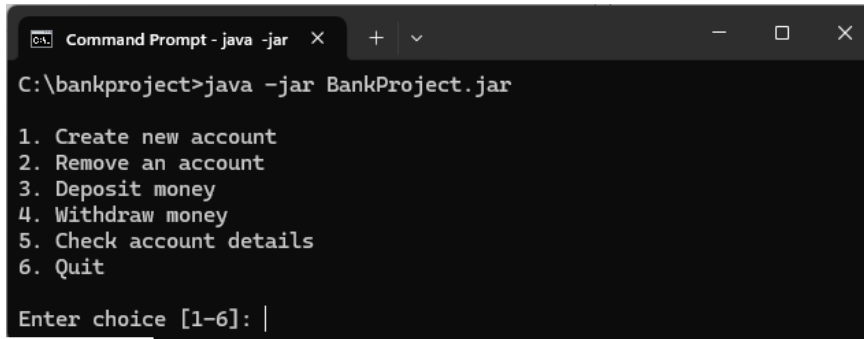
Figure 11

A file called `MANIFEST.MF` file will be included in the JAR file, and this will contain the information about which class is the main class.

We build and locate the JAR file as before (Figures 4 and 5). The resulting JAR file can now be run from the command line as follows:

```
java -jar BankProject.jar
```

This is shown in Figure 12



```
Command Prompt - java -jar
C:\bankproject>java -jar BankProject.jar

1. Create new account
2. Remove an account
3. Deposit money
4. Withdraw money
5. Check account details
6. Quit
Enter choice [1-6]: |
```

Figure 12

As you see, the argument `-jar` is used to show that we are executing a JAR file (here the file located in the current directory).

Running JavaFX programs from the command line

In the guide for working from the command line we used as an example the `RectangleGUI` class from chapter 17, which requires the presence of the `Rectangle` class.

Assume now that we have created an executable JAR file, `rect.jar` contained in a folder called `rect`. We need to set the module path to the directory which contains the JavaFX files that we have downloaded (`\JavaFX16\lib` in this case). We also need to add module components to the runtime environment as explained in the guides for using different IDEs. The command we need (in Windows®) is:

```
java -jar --module-path \javafx16\lib --add-modules javafx.controls,javafx.fxml rect.jar
```

This is shown in Figure 13.

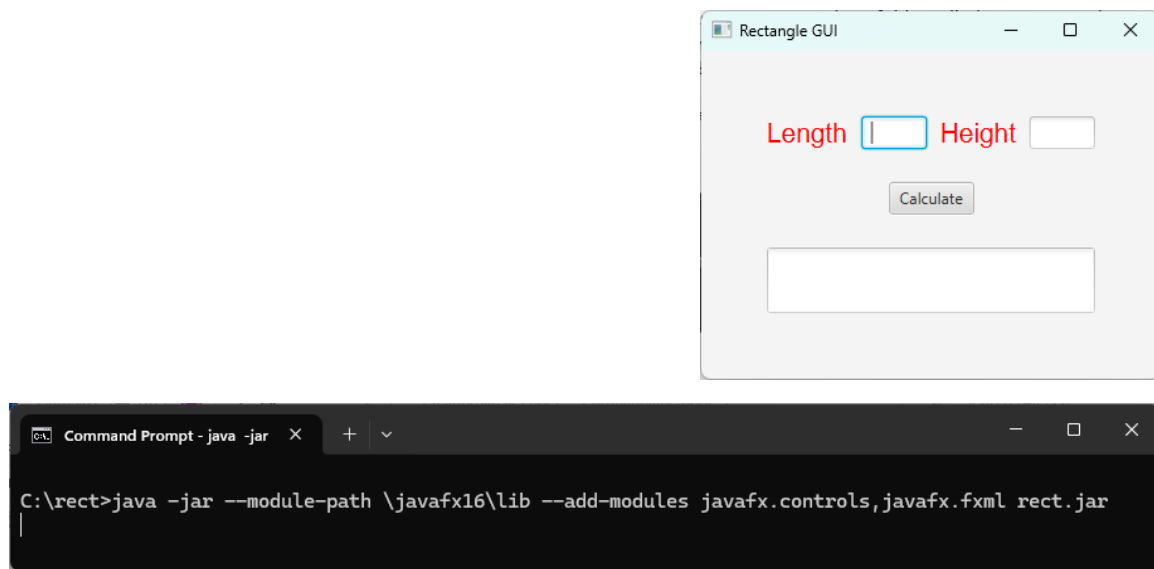


Figure 13

Double-clicking JAR files

Some of you may be aware that it is technically possible to run some JAR files by double-clicking them. However, we do **not** recommend this approach for the following reasons:

- **Console applications** close immediately after running, making it difficult to see the output.
- **JavaFX applications** require special runtime options (for example `--module-path`) which cannot be supplied via double-clicking.
- **Troubleshooting** is easier when you run your applications from a terminal or through an IDE, where you can see any error messages.

If you wish to run your programs by double-clicking on an icon, it is much better to write the command into a batch file (Windows) or a shell script (Linux®/Mac®). You can then create a shortcut to link to the batch file - if you don't wish to see the console, you can set the properties of the shortcut so that the console starts off as minimized.