

WEEK 1 LAB: CREATING OBJECTS AND PRACTICING JAVA SYNTAX

In this lab you will, in Task 1, create and manipulate some Java objects. In an object-oriented program objects often model some kind of real-world “things”. The objects in this lab represent shapes¹ (circles, rectangles, etc.). These objects can also draw a visual representation of themselves so that you can see the effect of anything you do with them. In Task 2 and 3 you will explore these objects further

You will also, in Task 4 and 5, practice writing Java expressions and statements. Which will help you to learn the syntax, or rules, of the language. Expressions and statements, which are the basic building blocks of any programming language. For this exercise you will use JShell which is an interactive tool for prototyping Java code, and is part of the Java platform.

You should attempt the steps for each task and try to answer the **questions** that follow some of the steps. It is recommended that you keep a lab note book as you work through the labs and take note of your answers to the questions in each lab.

Working with IntelliJ and lab code

You can install IntelliJ Community Edition in the lab from Apps Anywhere.

You may also want to install IntelliJ Community Edition on your own computer for use outside lab time. See <https://www.jetbrains.com/help/idea/installation-guide.html> for installation instructions

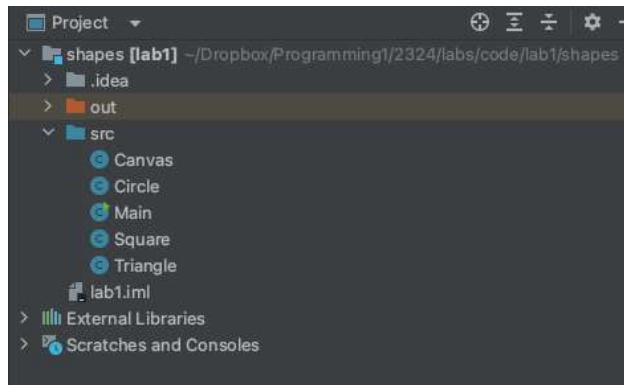
Most lab exercises will require you to download starter code from GCULearn. In most cases the code will be in the form of a ZIP archive containing one or more IntelliJ projects. Once you have downloaded the appropriate ZIP archive you can extract the contents by right-clicking and selecting **7-zip > Extract files...** from the pop-up menu. You can then start IntelliJ and use the **File>Open Project** option to navigate to the folder where you extracted the project and open it.

¹ Note that the starter code for Task 1 comes from a demo which is included with BlueJ, which is a useful Java IDE for learning

Task 1: Creating and manipulating objects

Getting started

1. Download *week1lab.zip* from GCU Learn and extract the contents to a suitable location
2. Start IntelliJ and select the **Open** option in the Welcome window to open the project *shapes*. You can find this project in the content you extracted from the ZIP.



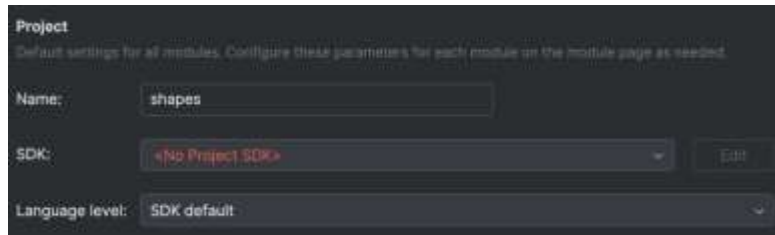
When the project is opened, the IntelliJ project window should look like the screenshot above. There are five **classes** shown.

3. Open the class *Main*. This should contain the following code:

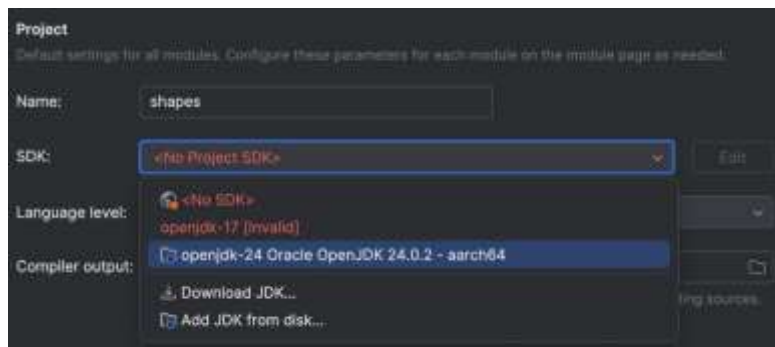
```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

This code defines a class which contains the *main* method, which is the starting point for a Java program and is where you initially write code to create objects. The default code is a basic “Hello World” program, which you can run now to test that everything is set up correctly.

When you first open the project you may see an **error message** or messages indicating that the **SDK is missing**. This may happen as the project that you downloaded doesn’t include the SDK that is required to compile and run the code. Go to the File > Project Structure menu item in IntelliJ and select the Project tab – this is where you can add an SDK to your copy of the project. In the following screenshot there is no SDK selected:



You can select an SDK from the dropdown list or choose the option in the list to download one. In the following screenshot, the downloaded project originally used openjdk-17, which is not installed on the current computer, but openjdk-24 is, so you would select the latter. If there is no SDK available, choose to download and select a version of openjdk – generally you can choose the most recent one.



- Once you have an SDK selected, click the green **Run** button in the top bar.



This will tell IntelliJ to compile all the classes in the project, and run the *main* method. Alternatively you can click on one of the green run buttons in the Main class editor window, and select run *Main.main()*. Check that the output window opens and you see the message *Hello World!*.

There should not be any syntax errors in the code at this point, but if there are, the compiler will report this to you. You are very likely to encounter errors when you start modifying the code, and you will need to correct these before the program will run. Be patient and think about what the error messages are trying to tell you.

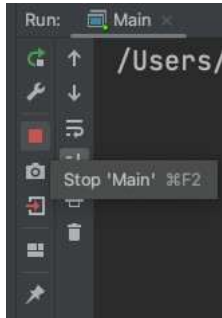
Creating objects

- Replace the *System.out.println* line with the following, making sure to type the semicolon at the end of each line. Note that Java is **case-sensitive**, so the lower and upper case characters matter!

```
Circle circle1 = new Circle();
circle1.makeVisible();
```

- Run the program again. This time you should see a new window open, separate from the IDE, with a circle drawn in it.

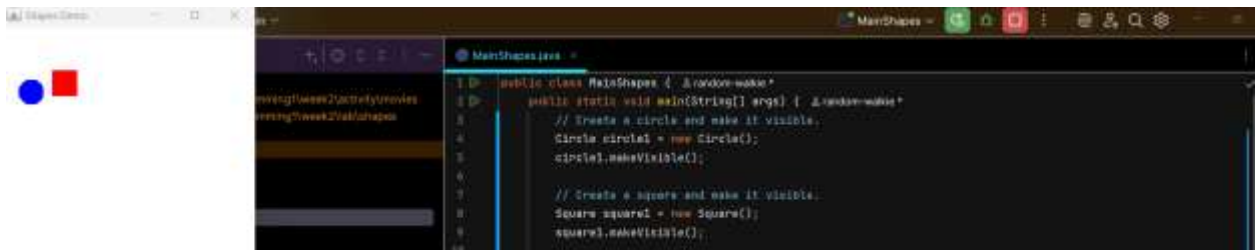
To close this window and stop the program, you need to click the red **Stop** button in the **Run** window at the foot of the IDE:



- Keep the existing code in *main*, and write additional code to create a **Square object**. Run the program again. Remember to **Stop** once you have seen the result.

Q1.1 What colour is the circle? What colour is the square? What key word in code is used when creating an object?

A1.1 The circle is blue, and the square is red. The keyword in the code used to create an object is **new**.

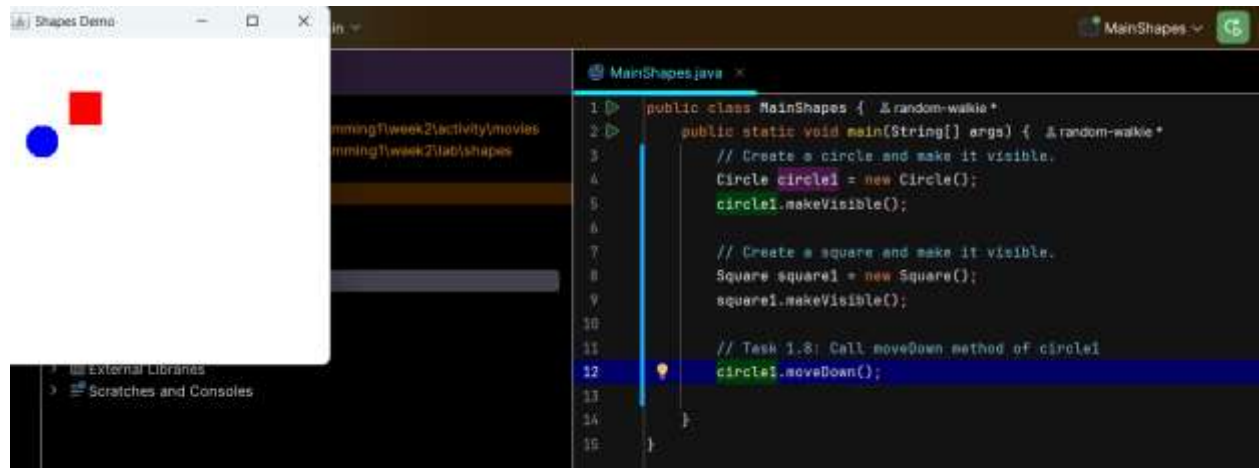


Calling methods

- Write additional code to call the *moveDown* method of *circle1*, and run the program again. Note that when you type *circle1* followed by a dot a list of available methods pops up so that you know what you can do with a circle object.

Q1.2 What difference do you see in the picture this time?

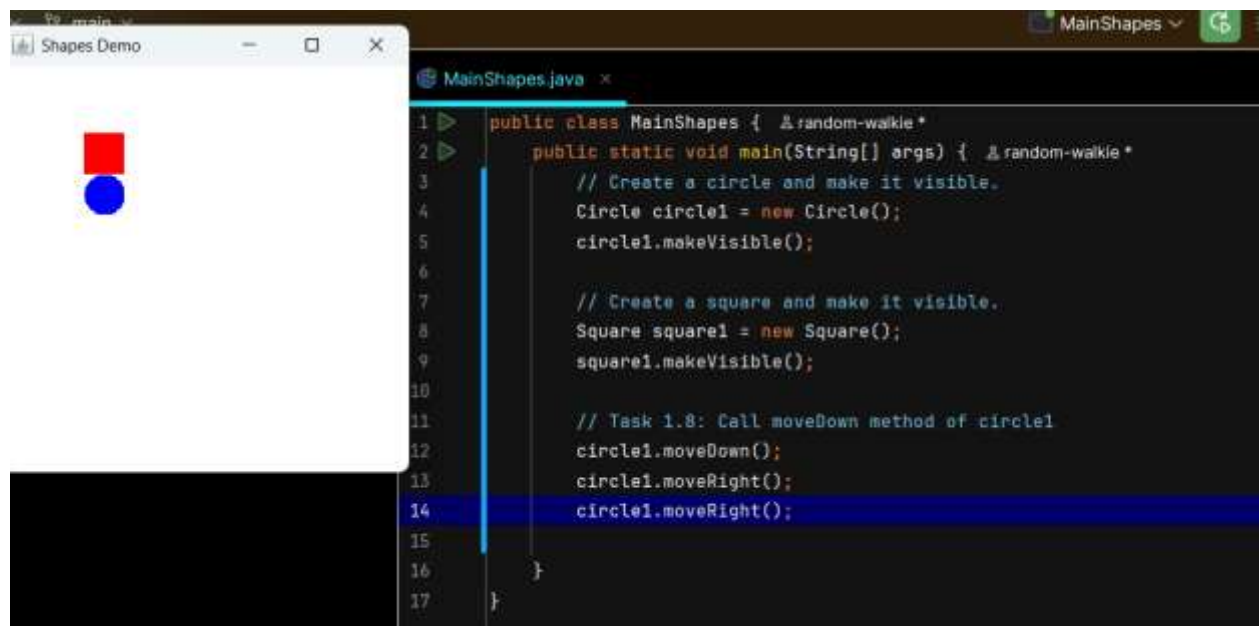
A1.2 The circle has moved down in the y axis of the displayed window.



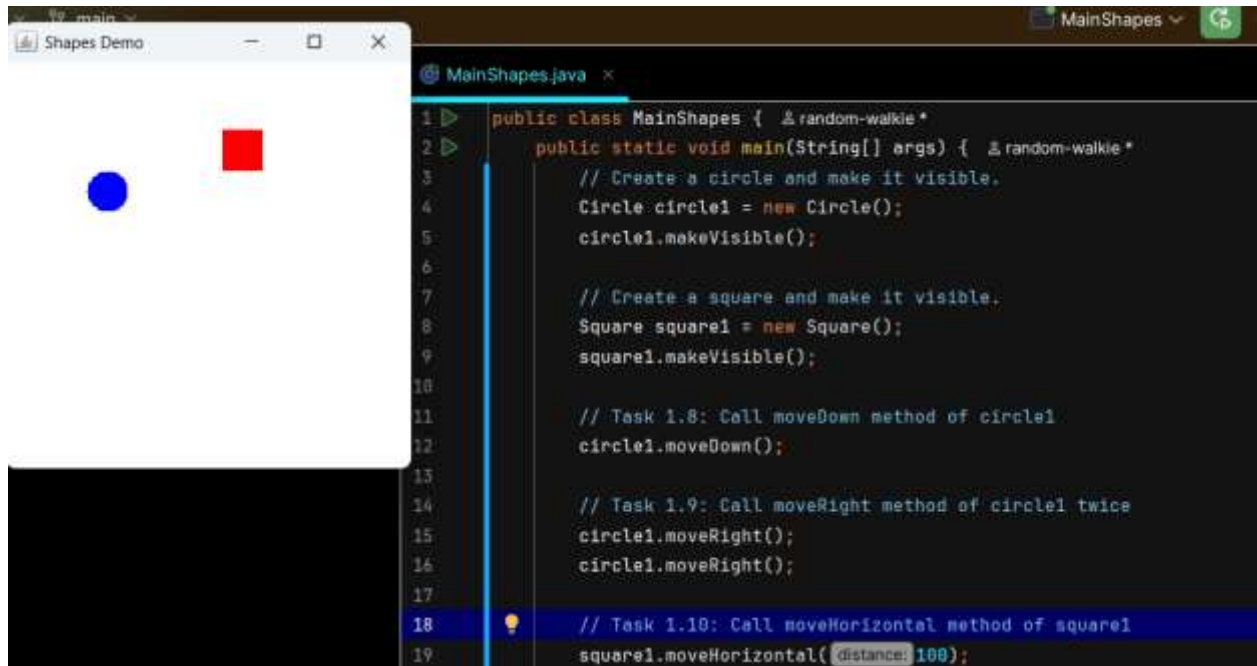
9. Call an additional method or methods to move the circle object so that it lines up directly below the square.

Q1.3 What method(s) did you use? How many times?

A1.3 I used moveRight() two times:

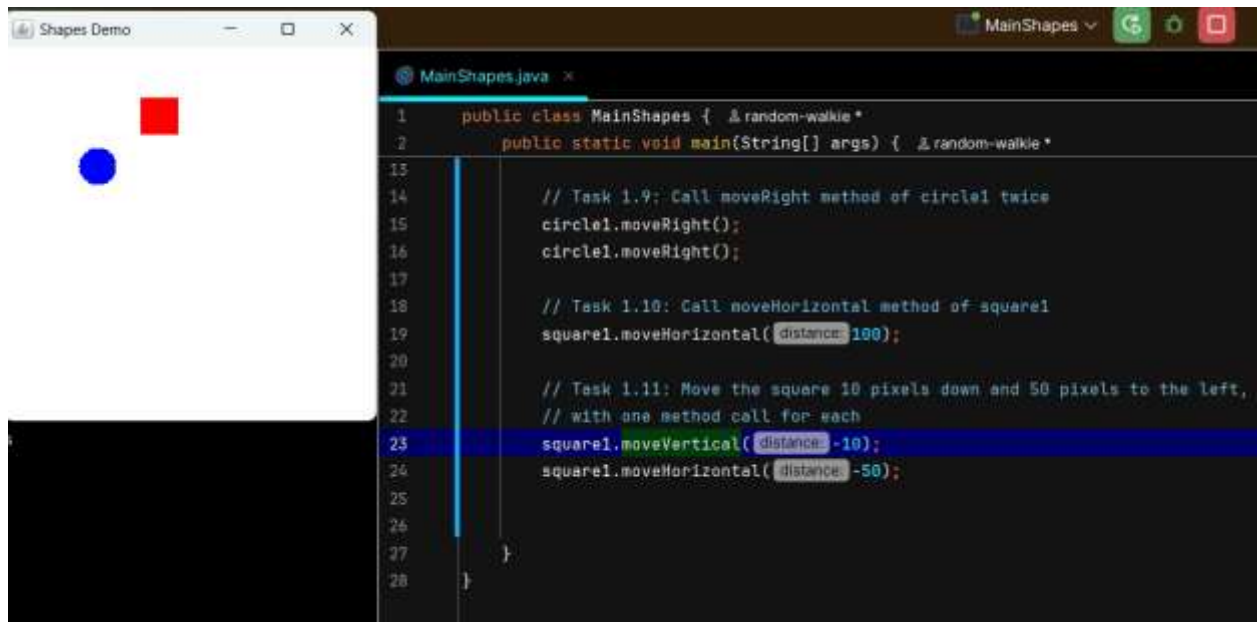


10. Call the *moveHorizontal* method of *square1*. This method needs some additional information – how far you want to move. Enter a value of 100. Observe the effect on the picture.

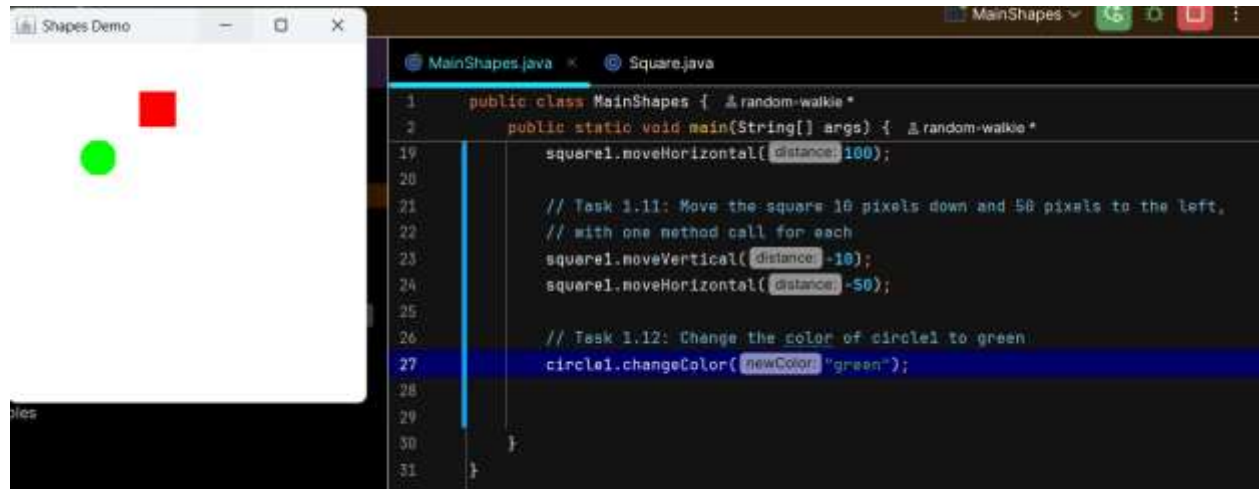


11. Move the square 10 pixels down and 50 pixels to the left, with one method call for each.

Q1.4 What method(s) did you use? How did you specify the direction (up or down, left or right)?



12. Call the *changeColor* method of *circle1* (note US spelling of *color*!). Enter the value "green" and observe the effect.



Q1.5 What is the data type of the parameter for this method call? What happens if you enter green, without quotes? What happens if you enter a number, again without quotes?

A1.5:

- The data type is String object.
- If I enter green without quotes, I get this error:

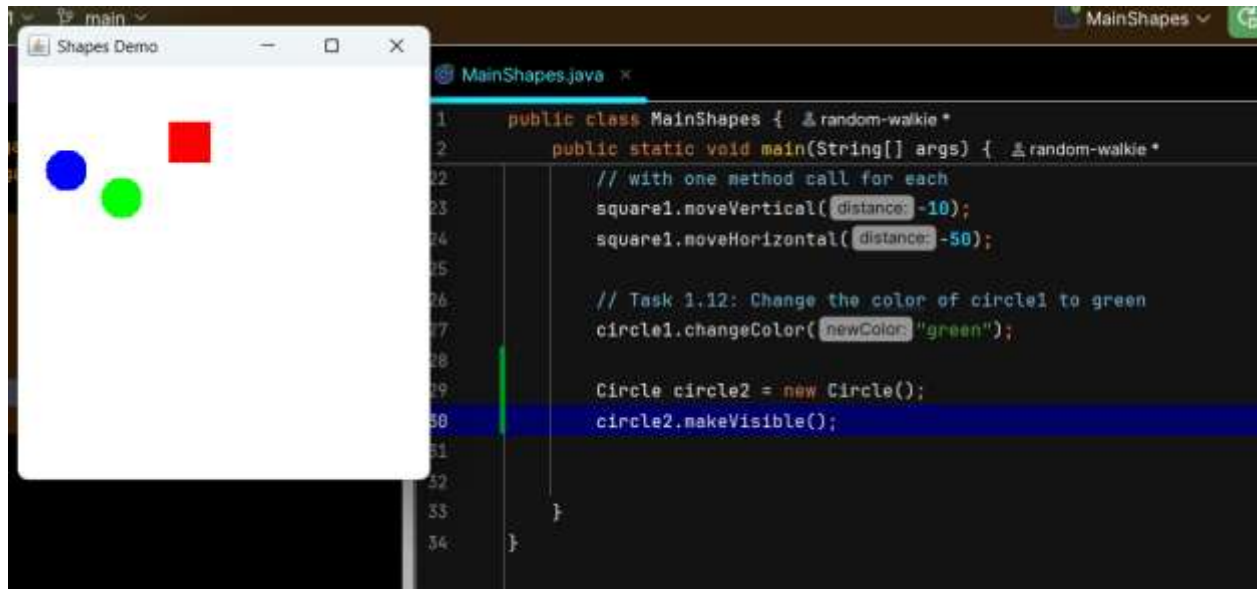
```
java: cannot find symbol
symbol:   variable green
location: class MainShapes
```

- If I enter a number without quotes:


```
java: incompatible types: int cannot be converted to
java.lang.String
```

Creating more objects

13. Write additional code to create another circle object, named *circle2* and make it visible in the picture.



Q1.6 How many Circle classes do you see in the project? How many circle objects? How many circle objects do you think it would be possible to create?

A1.6:

- I see 1 Circle class defined in Circle.java.
- I see two Circle objects (circle1 and circle2).
- As many as can fit in memory (until JVM runs out of memory!)

14. Inspect the objects by adding code to call `System.out.println` for each object. Look at the field names and field values.

Q1.7 What do the circle objects have in common? How do they differ? What makes a circle different from a square?

A1.7: The output from `System.out.println` for

circle1

DIAMETER: 30

XPOSITION: 60

YPOSITION: 80

COLOR: green

circle2

DIAMETER: 30

XPOSITION: 20

YPOSITION: 60

COLOR: blue

square1

SIZE: 30

XPOSITION: 110

YPOSITION: 40

COLOR: red

- The circle objects have same properties, with different values.
- Circle objects have a diameter attribute, whereas the square object has a size property.

Task 2: Working with interacting objects

Getting started

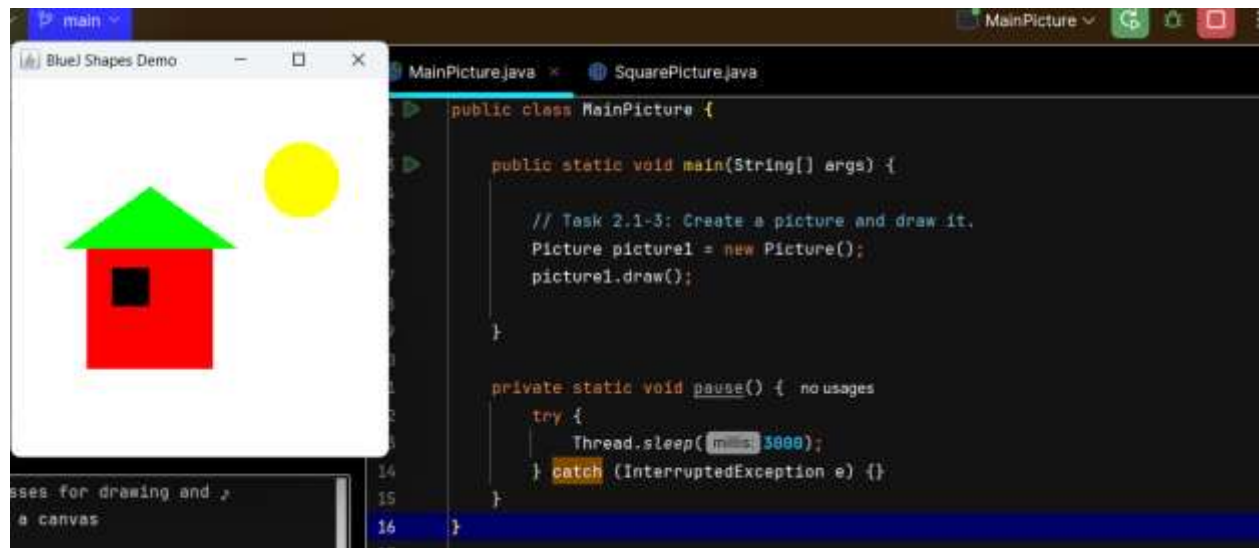
1. Close the *shapes* project and open the project *picture* which is also in the *week1lab.zip* download.

This project contains the same shape classes as the previous one, but there is also an additional class *Picture*.

Note: because I am organizing these projects in IntelliJ within the same directory, I had to do some code refactoring, otherwise we would have conflicts at compile time (classes can't have same, etc, etc).

Creating interacting objects

2. Find the *main* method in the *Main* class and add code to create an instance of the class *Picture* with name *picture1* in the Object Bench.
3. Add code to call the *draw* method of *picture1*, and run the *main* method. You should see a very simple picture of a house made up from simple shapes. These shapes are instances of the *Circle*, *Square* and *Triangle* classes you saw in Task 1.



Q2.1 How many objects, including the picture object are there? How many objects did you directly create? How do you think the other objects were created (have a look at the code in the *Picture* class)?

Calling methods

4. Add code to call the *setBlackAndWhite* method of the picture object and run the

code again.

Q2.2 What method does a square object provide to change its colour? What method did you call? How did the square's method get called?

5. Add the following line of code before the call to `setBlackAndWhite` and run the code again:

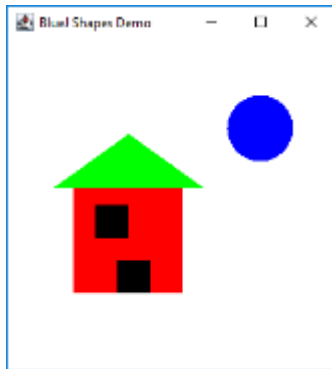
```
pause () ;
```

Q2.3 What did you observe? Can a picture be changed once it has been created?

Task 3: Modifying source code

In this final task you will try to make some changes to the Java code in the *picture* project and observe the effect of the changes. You will learn much more about how to write code as you progress through the module, but you can try this task now if you are feeling adventurous. Remember to stop the program after each step and to run the code again for each step.

1. Continue to work with the *picture* project. Open the class *Picture*. Look for the code that defines the colour of the sun in the picture. This code will be inside a block of code that defines a method called *draw*. Change the code so that the **sun will be blue** rather than yellow.
2. Compile the class by clicking the **Compile** button in the code editor, and **test your change** by creating a new picture object and calling its *draw* method, i.e. the method you have just modified. Note that every time you compile code, all existing objects are removed from the object bench and the picture window is closed.
3. Add a door to the house. This should be a square the same size and color as the window, and should be at the bottom of the wall, like this:



To do this you will need to add a new line:

```
private Square door;
```

immediately after the similar line which declares the window field, and then add code into the *draw* method to create and position the door. **Test your change** - it may take some trial and error to get the position right.

4. Add code at the end of the *draw* method to make the sun set. It should move slowly downwards until it disappears from the picture. *Hint*: use the method *slowMoveVertical* of *Circle*. **Test your change** by creating a picture and calling the *draw* method.
5. Separate out the code to make the sun set into a new method *sunset* so that it can be called at any time. **Test your change** by creating a picture and calling the *draw* method and then the *sunset* method.

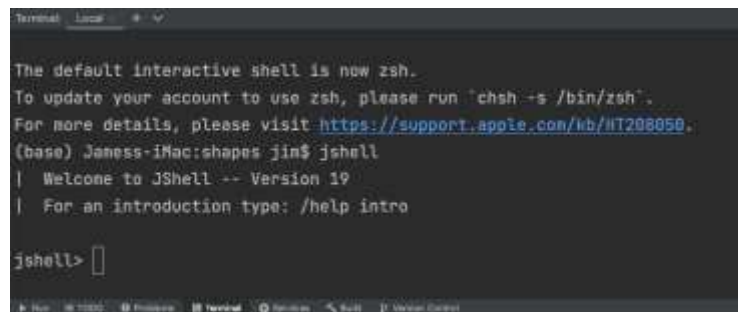
Starting JShell

The remaining tasks in this lab will use JShell. This section gives instructions on getting started with JShell for **tasks 4 and 5**. You can use JShell within IntelliJ if you have a Java project open.

1. Open the project from Task 1 if it is not already open. Click the **Terminal** tab at the foot of the IDE window to open a system terminal window. The screenshot below shows this on a Mac, the content of the terminal will look a bit different on other systems.
2. Enter the command;

```
jshell
```

at the terminal prompt. The **JShell** tool should start up and show the JShell prompt.



**If JShell doesn't start up as shown in the Terminal, it may be because the bin directory of the JDK, which contains the JShell executable, is not on the path. You may be able to add this directory to the path, or include the full path to the JDK bin folder in the command that you enter to start JShell. This path will depend on your OS and the way your JDK was installed. Examples:*

MacOS:

```
/Users/myname/Library/Java/JavaVirtualMachines/openjdk-24.0.2/Contents/Home/bin/jshell
```

Windows:

```
C:\Users\myname\.jdk\openjdk-24\bin\jshell
```

You can instead use the JShell Console tool in IntelliJ, which you can open with the Tools / JShell Console menu option. This tool has quite a different user interface, which you may or may not like.

As a further alternative, you may be able to find an online JShell environment. At the time of writing, <https://tryishell.org/> was available.

You are now ready to start typing Java statements. If you enter an expression, JShell will immediately evaluate the expression and show the result with the value assigned to a default variable name, e.g.

```
jshell> 20 / 10  
$1 ==> 2
```

If you type a statement that assigns a value to a variable, then the variable name and assigned value are shown, e.g.

```
jshell> int x = 10;  
x ==> 10
```

You can get more detailed output from JShell commands if you first run the command:

```
/set feedback verbose
```

To close JShell and return to the terminal prompt you can enter `/exit`.

3. Create a new Word document and save it as *lab2_answers*. You should record in this document information that you are asked to take note of and your answers to questions asked in the exercises.

Task 4: Expressions

Arithmetic expressions

In the first exercise you will write some expressions that perform operations using **arithmetic** operators. The operands in these expressions are simply numeric values. Review the presentation and make sure you understand the purpose of each operator.

1. Enter the following expressions in JShell - type in each expression, and press the return key after each one. Take a note of each expression and the **type** and **value** of the result.
 - A. `10 + 20`
 - B. `10 + 20 - 3`
 - C. `50 - 10 * 3`
 - D. `(50 - 10) * 3`
 - E. `20 / 10`
 - F. `10 / 20`
 - G. `10.0 / 20`
 - H. `30 % 20`

Are the results as you would expect? Why does expression C give a different result from D? Why does G give a different result from F?

Expressions with variables

In this exercise you will declare some **variables** and write some expressions that perform operations on these using arithmetic operators.

2. Enter the following variable declaration and initialisation statements in JShell – these variables will be used throughout Task 2, so if you need to stop and start again part of the way through the task you will need to enter these again before continuing.

```
int a = 5;  
int b = 10;  
int c = 20;  
int d = 20;  
boolean x = true;  
double y = 10.0;
```

3. Now you will evaluate expressions using these variables. Enter the following expressions. Take a note of each expression and the **type** and **value** of the result. Are the results as you would expect?

- A. $a + b + c$
- B. $a * c - b$
- C. $a * (c - b)$
- D. b / c
- E. y / c

4. Change the value of *a* to 30 using the following assignment statement:

```
a = 30;
```

Enter each expression in part 3 again, note the value of the result in each case and compare to your answers from part 3. Which ones did not change value, and why?

Change the value of *a* back to 5 using an assignment statement.

5. Enter the following declaration statement using *var* instead of an explicit type.

```
var aa;
```

What happens, and why?

Enter the same set of variable declaration and initialisation statements as in step 2, except with variable names *aa*, *bb*, etc. and using *var* for type inference. Check that the variables are given the types you expect.

Conditional expressions

This exercise should help you to understand conditional expressions, which apply **relational** and/or **conditional** operators and evaluate to *boolean* values. Review the presentation and make sure you understand the purpose of each of the operators.

6. Enter the following expressions. Take a note of each expression and the **value** of the result.

- A. `a < 10`
- B. `a <= 10`
- C. `c == d`
- D. `c == b`
- E. `c != d`
- F. `x`
- G. `(a==10) && x`
- H. `(a < 10) && (c == d)`
- I. `(a < 10) || (c == d)`
- J. `(a > 10) && (a < 20)`
- K. `(a >= 10) && (a < 20)`

Are the results as you would expect?

7. Now devise and test expressions that will test for the following conditions, which for the variable values assigned in step 1 should all be *true* (as long as you remembered to set *a* back to 5 in step 4). Take a note of the expression you devise for each condition. In each case, there may be more than one possible way of achieving the required result.

- A. *a* is between 0 and 6 (that is, in the range 1,2,...,5)
- B. *b* is not between 0 and 5
- C. at least one of *a* and *b* is equal to 10
- D. of the values of *a*, *b* and *c*, *a* is the smallest and *c* is the largest

Expressions using the increment operator

8. Enter the following expressions, in the order shown. Press the return key after each and take note of the resulting value of each expression. This should help you to answer the question in the following step.

- A. `a`
- B. `--a`
- C. `a--`
- D. `a++`
- E. `a`

9. What expression would evaluate the current value of a variable *myVar* and then decrement the variable's value? Test your expression and take a note of your answer.

Task 5: Statements

1. Continue with the variables you declared in Step 2 of Task 4. Re-enter these declarations if you have closed JShell since you did Task 4 and the variables are no longer available in the terminal.
2. Enter the following **expression**, and then evaluate the variable *a* by entering another expression which is simply the name of the variable.

```
a * 2
```

Enter the following **statement** (note that a statement ends with a semi-colon).

```
a = a * 2;
```

What output do you see? Now evaluate the variable *a* again. Note that an expression does not assign or change the value of variables, but a statement can.

3. Enter the following statements in turn. For each one, note whether it assigns a value to a variable (you can evaluate variables to check), and if not, what effect it does have. Where a value is assigned, evaluate the variable as an expression and note the value.

- A. `int e = b * c;`
- B. `b += 20;`
- C. `b *= b;`
- D. `c *= c + 5;`
- E. `a--;`
- F. `double z = b / 20.0;`
- G. `System.out.println("Hello");`
- H. `System.out.println(a);`

4. Now devise and test statements that do each of the following. Take a note of each statement that you devise and the result of your test.
 - A. Declares a new *double* variable *varA* and initialises its value to 2.5
 - B. Subtracts 2 from *c* using a shortcut assignment operator
 - C. Declares a new *int* variable *varB* and initialises its value to the product of *a* and *c*
 - D. Outputs the sum of *a* and *b* to the terminal window