COM1027 Programming Fundamentals

Lab 6

Introduction to enums

Purpose

The purpose of this lab is to develop your skills and ability to:

- Defining your own simple enumerated types.
- Defining your own enumerated types with data and behaviour.
- Using enums as arguments to methods and relating the arguments to underlying data structures.

Lab Structure

Labs are a mixture of step by step instructions that enable you to learn new skills, and exercises so that you can define your own examples.

We will use **Concept Reminders** in front of text when we are reminding you of concepts you have already learnt previously in labs or lectures.

In using this document it is possible to paste text directly into Eclipse. That means when defining pieces of code contained in lab sheets in Eclipse their value can be copied directly from here, which will remove the possibility of mistyping values. If you are reading a PDF version of the document, then clicking on the 'Select' icon activates text selection. Before running the code, ensure that the code has been copied correctly, and make the relevant changes to make the code fully-functional.

These lab exercises will be **assessed**. They are marked, and contribute to your final grade. This lab exercise has 10 marks to earn which will contribute towards your final grade. Every place you have to add code is indicated as an 'Exercise' with instructions.

A total of 10 labs will be assessed, that will correspond to the first 20% of your final grade. Please submit your completed work using GitLab before 4pm on Friday Xth Month 2021.

You must comply with the University's academic misconduct procedures: `https://www.surrey.ac.uk/office-student-complaints-appeals-and-regulation/academic-misconduct`

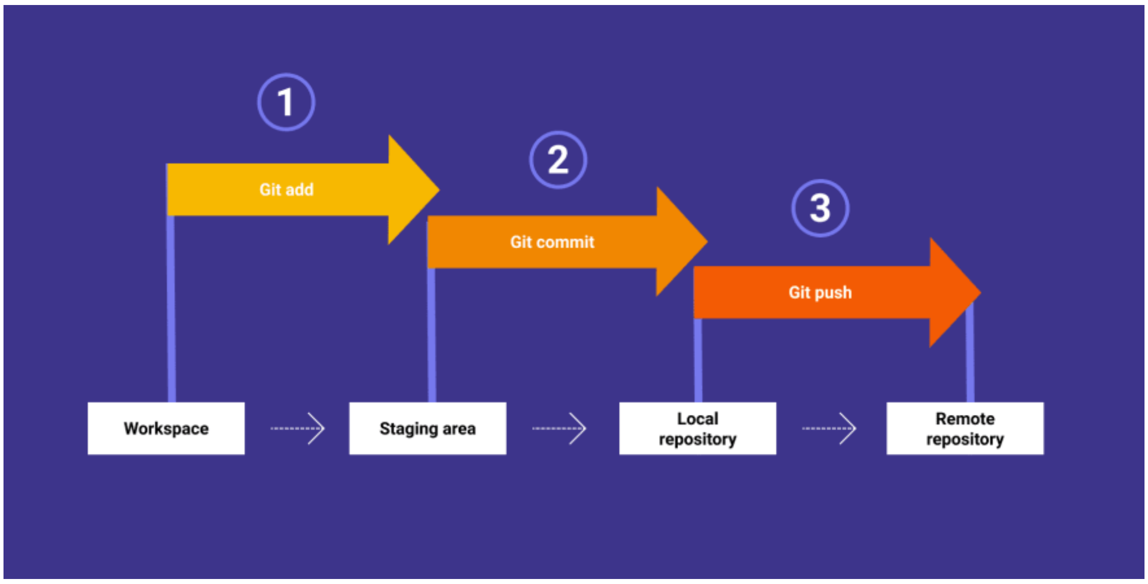
Please ask us lots of questions during the labs, or use the discussion forums on SurreyLearn. All the demonstrators are here to help you get a better understanding of the Java programming concepts.

Instructions

For this module, we will be using automated assessment tools for all lab activities and coursework. To achieve this, we will be using the Faculty of Engineering and Physical Sciences (FEPS) GitLab platform to upload completed code.

In last week's lab activity, you completed the following:

- Downloaded a project from SurreyLearn and imported it into your workspace
- Made changes to the project by creating new classes
- Used the UML diagrams to convert the structured English language to Java code
- Commited the changes to your local repository and
- Pushed those changes onto your remote repository on GitLab



Screenshot taken from `https://docs.gitlab.com/`

In this lab, you will download a new project from SurreyLearn, make substantial changes to the project, commit and push those changes to your remote repository, just like in previous weeks.

Important Note: Do not push any buggy code to the repository unless you have to. Use the `'Run As > Java Application'` or `'Run As > JUnit Test'` as indicated in the instructions below.

Reminder: You can access your GitLab repository via `https://gitlab.eps.surrey.ac.uk/`, using your username and password.

Example

In this example, we will use the debugging tool to inspect instance variables and see in detail how the values change in each line of code.

1. Import an existing Eclipse project and set your workspace
From SurreyLearn, download the project called `COM1027_Lab06.zip`. Extract the contents of the project in the Downloads folder. Copy the uncompressed (unzipped) `Lab05` folder to your local repository. **Note:** Do not copy the `COM1027_Lab06` folder. Navigate in the folder, and only copy `Lab06` folder.

If you are using the computer labs, then your local git directory would look like this:
`/user/HS223/[username]/git/com1027[username]/`
For example:
`/user/HS223/sk0041/git/com1027sk0041/`
where sk0041 shows a sample username.

Start Eclipse and open the same lab workspace as last time. Import the project in Eclipse, by clicking on `File > Import > General > Project from Folder or Archive`. Select `Directory` to locate `Lab06` from your local git repository. Once the files are loaded, select the Eclipse project and click `Finish`.

2. Simple enumerated types

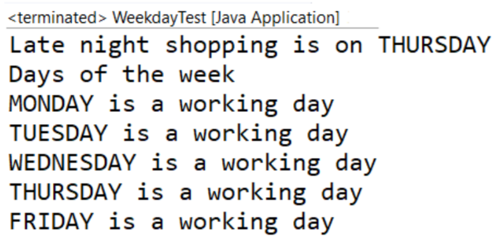
Once imported, look at the `Weekday` enum class in the `lab6_example` package. You will see that a simple enumerated type consists of just a list of the names that the type represents.

Concept Reminder:
By convention, we always use UPPERCASE for each enumeration constant. This distinguishes a enum wherever we use the values.

Open the `Example1` class. You will see that the aim of this file is to assign a particular day to be the late night shopping day. This shows you how enums can be used in other classes. Note that we do not need to use `new` to use an enum value.

This method also shows how we can output the value of an enum automatically (each enum has an automatic `toString()` method which turns the enum value into a String). We can also iterate over all of the values in an enum using a `foreach` statement.

Run the program and you will see:



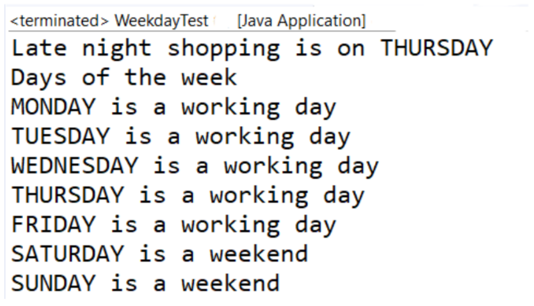
```
Late night shopping is on THURSDAY
Days of the week
MONDAY is a working day
TUESDAY is a working day
WEDNESDAY is a working day
THURSDAY is a working day
FRIDAY is a working day
```

In fact there is no need to declare the `daysArray` variable. We can simply use the `Weekday.values()` inside the `foreach` loop. Change the code to do this.

Add the missing values for Saturday and Sunday to `WeekDay`.

Now that we have added Saturday and Sunday the output from the code is wrong because Saturday and Sunday are now listed as a working day.

Add case clauses in the `switch` statement to enable the following to be printed:



```
Late night shopping is on THURSDAY
Days of the week
MONDAY is a working day
TUESDAY is a working day
WEDNESDAY is a working day
THURSDAY is a working day
FRIDAY is a working day
SATURDAY is a weekend
SUNDAY is a weekend
```

Run the code again, by right-clicking on the `Example1` class and select `Run As > Java Application`.

Add the following line of code to the bottom of your `Example1` class and run the code again.

```
System.out.println(Weekday.FRIDAY.ordinal());
```

Explanation: The ordinal() method is provided to extract the order number of an enumeration constant (starting at 0).

This method is *often abused* because it is used to relate positions of enumerated constants with array positions. You should **BE CAREFUL** before using this in practice.

3. More complex enumerated types

Typically the above example also contains constant information, the fact that Monday to Friday are working days is fixed in the UK. So is there a way to incorporate this into the enum definition itself? Yes.

Look at `Weekdays` enum class. You will see that the enumeration constants now have parameters after each name place in brackets, such as:

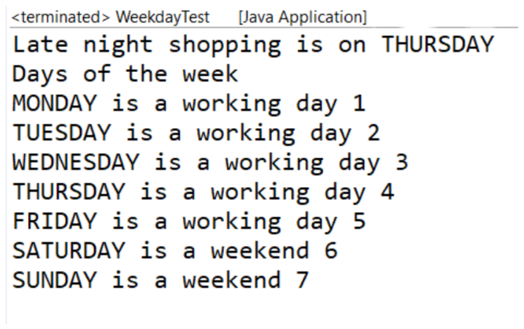
```
TUESDAY("working day")
```

The other notable differences are:

1. Inclusion of a field which stores the day type.
2. Inclusion of a **PRIVATE** constructor. The fact that this is private is important because we will not be making objects of the enum class. The enum constants are themselves objects.
3. Inclusion of a getter to extract the day type information related to a particular day.

Run the `Example2` program in the same package as a `Java Application`. This time you will see that `day.dayType()` is being used to retrieve the fact that it is a working day or a weekend. (This method call gives you a clearer picture that enumeration constants are themselves objects because day is the object and `dayType()` is the method invocation. Go back to the `Weekdays` enum class and change it to also include the fact that Monday is day 1, Tuesday is day 2, and so on. This will mean that you need to:

1. Include an extra parameter in the constant definition, for example: `Monday("working day ", 1)`
2. Add another field and call is `dayNumber`
3. Add another parameter to the existing constructor and assign the field
4. Add another appropriate getter
5. Go back to the `Example2` class and change the code to achieve the following output:

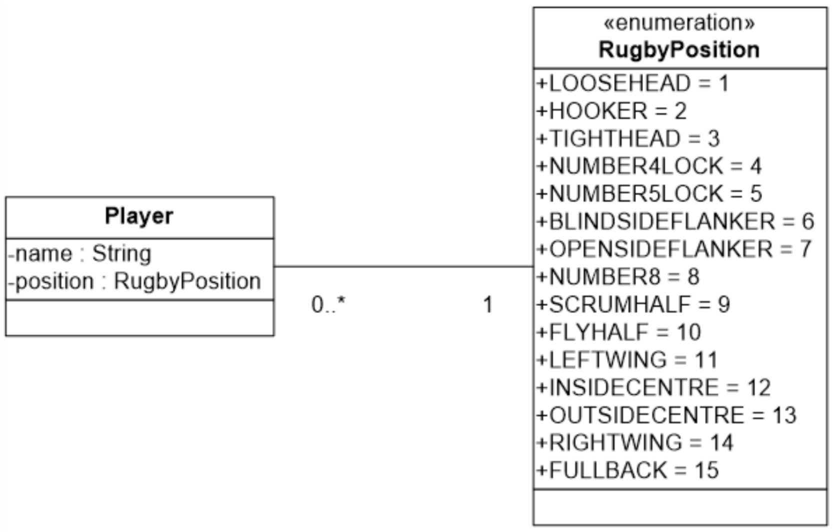


```
Late night shopping is on THURSDAY
Days of the week
MONDAY is a working day 1
TUESDAY is a working day 2
WEDNESDAY is a working day 3
THURSDAY is a working day 4
FRIDAY is a working day 5
SATURDAY is a weekend 6
SUNDAY is a weekend 7
```

Exercise 1 (2 Marks)

This exercise requires you to create a new class which will make use of enums and repetition statements.

First, ensure that the code for this exercise is added in the `lab6_exercise1` package. Define an enum class called `RugbyPosition` and a normal concrete class `Player` class as per the class diagram:



The notes below can help you structure your code:

- In your enumeration you must include parameterised enumeration constants together with the appropriate field, and constructor. This is to enable `Player` objects to be created as follows:

```
Player player = new Player("Rhys Priestland", RugbyPosition.FLYHALF);
```

- Add getters for each field. Note the return type of the position getter will be the enumerated type RugbyPosition.
- Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

Exercise 2 (3 Marks)

This exercise requires you to reuse the classes used in Exercise 1 using your understanding of enums and data structures.

The code for this exercise, will need to be added in the `lab6_exercise2` package. Firstly, copy the `Player` class and `RugbyPosition` enum and add them in the `lab6_exercise2` package.

Define a class called `Halfbacks`. In rugby, two players form the halfbacks and so we will use an array made up of two elements of type `Player` to hold this information.

Important Note: We realise that this is artificial. With just two elements you could easily store them as separate fields but we want to store them in an array because we want to have practice relating enumeration values to underlying data structures.

- Define this field in the class and provide a constructor which simply creates an empty array object.
- Define a method called `assignHalfback` which takes a player object. If the player is a scrumhalf it is assigned to the first element of the array and if it is a flyhalf it is assigned to the second element of the array. Otherwise the assignment does not happen. It is the developer's responsibility to make sure that the relationship between the data structure and the enum values makes sense and is well defined so that the code can be maintained.
- Define a new method called `displayElements` which loops through the array and returns a String representation of the array's contents, in the following format:

  ```
  0 - Mike Phillips SCRUMHALF
  ```

  OR

  ```
  0 -
  ```

  if the player is not a scrumhalf or flyhalf.

  To achieve this output, make use of a `StringBuffer`. No changes are to be made in the `Player` class.

- Define a new class called `Check` which will enable you to check the functionality of the array (initialisation and display). The new class contains the following code in a main method:

  ```java
  Player player1 = new Player("Rhys Priestland", RugbyPosition.FLYHALF);
  Player player2 = new Player("Mike Phillips", RugbyPosition.SCRUMHALF);
  Halfbacks halfbacks = new Halfbacks();
  halfbacks.assignHalfback(player1);
  halfbacks.assignHalfback(player2);
  System.out.println(halfbacks.displayElements());
  ```

The next thing we want to do is test that who is playing at flyhalf, we know it is Rhys Priestland but we should be able to extract this from the data structure. The following code is **REALLY BAD**.

- Include it in the `Halfback` class because it is important to see what it does:

```java
/**
 * Really bad way of extracting detail from an internally managed
 * array. We should never ask for the array index as this is
 * implementation detail.
 *
 * @param number index into array - DO NOT DO THIS
 * @return the name of the player at the specified array index.
 */
public String getPlayerBad(int number) {
  String result = null;
  if (number == 0) {
    result = this.halfbacks[0].getName();
  }
  if (number == 1) {
    result = this.halfbacks[1].getName();
  }
  return result;
}
```

- In the `Check` main method (under the other code you have already included) add the following line and run the code.

```java
// Not a good way of extracting the player...
System.out.println(halfbacks.getPlayerBad(1));
```

Explanation: this code makes the developer know that the value 1 is representing the value of the player in the array he is looking for. It would be much better to allow the developer to provide the information for `RugbyPosition.FLYHALF` and the code in `Halfback` work to relate the parameter value to the data structure.

- So instead of providing a method which makes explicit reference to the data structure, you should now define a method with the following signature:

```java
public String getPlayer(RugbyPosition position)
```

This tests the value of the positions and if it is a flyhalf or a scrumhalf it returns the appropriate string otherwise it returns `null`. This requires to make use of a `switch` statement similar to the `assignHalfback` method.

- Include a line of code in the `Check` class which confirms your understanding of who is playing at flyhalf.

- Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

Explanation: The purpose of this exercise was to relate data structures with objects which make use of enumerated types. It is important to know that even though enums have a natural ordering of their own because they appear in the list, this list is not always going to be the order you want to have your data structure. (So you should think twice before using `.ordinal()` because it is not good style.)

Stretch & Challenge Exercise (5 Marks)

This exercise requires you to create a new program, and a new set of classes.

First, make sure that you use the correct package for your code - `lab6_exercise3`.

The aim of this exercise is to represent a table that shows the number of medals each country has won.

Consider some of the data from the medal table of the Olympics in 2020 (fake data):

	BRONZE	SILVER	GOLD
BRITAIN	0	0	5
CANADA	2	0	0
CHINA	1	2	3
RUSSIA	3	0	1
FRANCE	1	0	0

- Define two enum classes for `Medals` and `Countries`.
- The two enums are required to have a set of paramaterised enumarated constants. For example, `Countries` are represented with two values `CANADA (1, false)`, where `1` represents the index of the country and `false` shows that the country is not within EU. The `Medals` are represented with only one value, for example `BRONZE (0)` where `0` represents that index of the medal *(Note: Silver can be set to 1, and Gold to 2)*
- Each enum has a `getIndex` method that returns the index of the enumerated constant as per its attributes.
- Define a `MedalTable` class which contains a double array which is a data structure to hold the medal table info. The data held in this array is of type `int`.
- Consider adding a default constructor in the `MedalTable` class that sets the size of the array using the following statement:

```
this.medals = new int[Countries.values().length][Medals.values().length];
```

- A new method called `addMedal` needs to be defined that would enable you to initialise the values of the array, taking into account the `Countries` and `Medals` values within its parameters.
- Define a `Check` class which creates an object of `MedalTable` and initialises it to the above table. Use this class to test the functionality of your code.
- Define a method in the `MedalTable` class which takes in a country and a medal colour and outputs the appropriate number of medals.
- Extend the `Countries` enum to identify whether the country is in the EU or not.
- Add a method to `MedalTable` which determines the total number of medals the EU countries have gained. No arguments need to be passed into this method.

- Revisit the `Countries` enum and add the following method - What does this code do?

```java
/**
 * Method to return the countries in the correct order.
 *
 * @return an array of the countries in the correct order.
 */
public static Countries[] orderedCountries() {
    // Use the index as an array index.
    Countries[] output = new Countries[Countries.values().length];

    for (Countries country : Countries.values()) {
        output[country.getIndex()] = country;
    }
    return output;
}
```

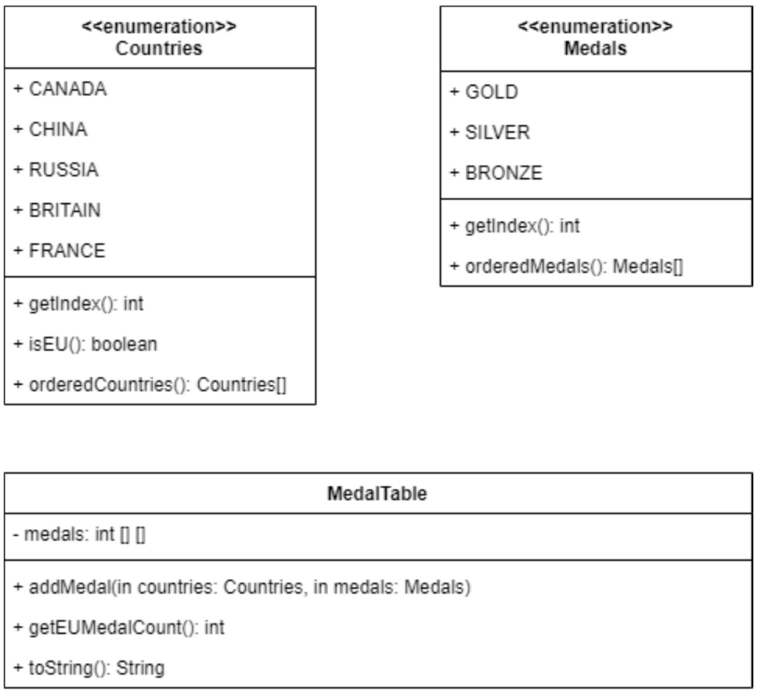
- Similarly, add the following method in the `Medals` enum:

```java
/**
 * Method to return the medals in the correct order for the table.
 *
 * @return an array of the medals in the correct order.
 */
public static Medals[] orderedMedals() {
  // Use the index as an array index.
  Medals[] medals = new Medals[Medals.values().length];

  for (Medals medal : Medals.values()) {
    medals[medal.getIndex()] = medal;
  }
  return medals;
}
```

- Modify the `Check` class to ensure that all the methods defined function as expected.

The complete UML class diagram below, shows the attributes and methods of the new enums and concrete class:

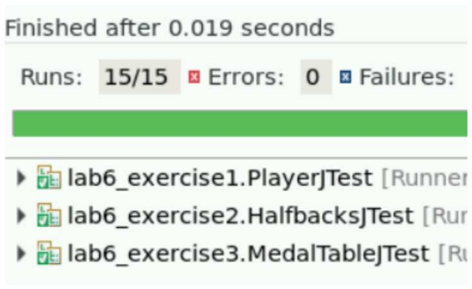


Test your code by right-clicking the project name and then `Run As > JUnit Test`. JUnit tests have been predefined in the `src/test/java` source folder. Do not modify them as this would cause your Maven Test to fail!

Final Steps

The final step of the lab activity is to check on whether the code is fully functional. To do this, right-click on the project name and select `Run As > JUnit Test`.

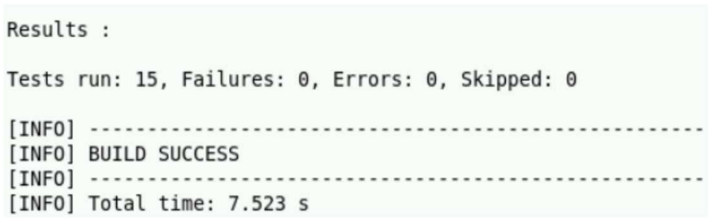
By selecting `Run As > JUnit Test` all the pre-defined JUnit tests will be called. You should get the following in order to proceed:



```
Finished after 0.019 seconds
Runs: 15/15   Errors: 0   Failures:
lab6_exercise1.PlayerJTest [Runner
lab6_exercise2.HalfbacksJTest [Rur
lab6_exercise3.MedalTableJTest [Ru
```

It is now time to test the structure of your Maven project. To do this, right-click on the project name and select `Run As > Maven Test`.

This should return the following message on the console:

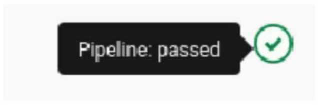


```
Results :

Tests run: 15, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 7.523 s
```

The test passes - now what?
You can now commit and push all the changes made to the remote repository. To do this, select the `Git Staging` view via the menu `Window > Show View > Other...` and then `Git > Git Staging`. Add all the unstaged changes to your local repository, and commit and push the changes to the remote repository.



Pipeline: passed