Lab 2. Generic Types

Theme. In this lab, you will:

- practise the use of type variables
- practise the use of generic types
- revise the use of enhanced for statements
- use inheritance to implement a flexible OOD
- practise method overriding
- use a text-based user interface to perform data input using the keyboard

Key concepts: generic types, type variables, arrays and collections, iteration, enhanced `for` statements, inheritance, `java.lang.Iterable`, `java.util.Iterator`, `java.util.HashMap`, `java.util.Scanner`

Required file(s): `lab2.zip`

1 Getting Started...

1. Download the archive `lab2.zip` from Blackboard. Extract the contents into your default `eclipse` workspace for this module, i.e. the folder `CS2310\Labs\`.
2. Start up `eclipse`.
3. Create a new Java project named **lucky_draw** using the contents extracted from `lab2.zip`.

2 Using Generic Types

The Cancer Research Foundation plans to run a raffle for fund raising. An application programmer was employed to implement a Java application for running the raffle. This Java application has a text-based user interface (TUI). It was intended to make use of a generic class `Box` to model the behaviours of a *prize box* and a *sold-ticket box*.

Note that this basic version of the application is **not** required to handle the money raised by the sale of raffle tickets.

Unfortunately, before the programmer completed his tasks, he left the Cancer Research Foundation for a better paid job. Your task is to complete the remaining Java classes according to the requirements.

Hint: The approximate locations where you are expected to add your Java code and relevant hints for accomplishing the tasks have been marked throughout the given Java programs. Look out for **block comments** that include a sequence of *four* exclamation marks, i.e.:

```
/* !!!!  ...  ...  */
```

Your Tasks

1. The class `Box` is intended to be a *generic* class which uses an `ArrayList` object to model its contents. Modify the definition of class `Box` by introducing the use of the type variable `T` in class `Box`.

> **Hint**: Do not start removing the syntax errors in class `Box` right away. Those syntax errors will be gone when you have completed the tasks described by the `/* !!!!` instructions.

2. An enhanced `for` statement is expected to be used to iterate through the contents of a `Box` object. To enable this, class `Box` needs to implement a certain Java interface. Introduce this required realisation relation into the definition of class `Box`.

3. A `Box` object should be able to reveal if it is empty. The header of the method `isEmpty` has been defined in the class `Box`, but its method body has not been completed. Finish off the implementation of this method by adding a single `return` statement.

4. Class `Prize` inherits the method `toString` from its super-class `Object`. However, the inherited implementation is not suitable to be used to reveal the name and value of the prize during the announcement of winners. Override the method `toString` in class `Prize` by supplying a suitable implementation of this inherited method.

5. Class `Raffle` models the typical behaviour of a raffle. It uses two `Box` objects to model a box of prizes and a box of sold raffle tickets. Each raffle ticket has a number and the name of its buyer. The implementation of class `Ticket` is complete, but the implementation of class `Raffle` needs a fair bit of work.

   Complete the required implementation for Class `Raffle`.

6. Class `TUIRaffle` simulates the process of running a raffle. Method `results` simulates the process of carrying out the lucky draw and announcing the winners.

   Write Java code to model the process of announcing the results.

3 Testing

Now test your implementation to see if it meets the above requirements. The top-level class for this project is `TUIRaffle`. You will find a static method with the signature `main(String[])` in it. Whenever you see that a class has such a `main` method, it means that you can execute the class as a Java application. You may test the application by using the **Run** drop-down menu in `eclipse`. Unlike class `GUIConference` in Lab 1, `TUIRaffle` does *not* require any run time argument.

Hint:

- Select the menu option `Run` ➠ `Run As` ➠ `Java Application`. See Lab 1 for detail.
- To compile and run the Java application without using any IDE, i.e. run it under the command line based interface (shell), the command for compiling your Java programs in a package is (assuming that your current folder is the project folder and that you are using Linux):

  ```
  javac game/*.java
  ```

  where `game` is the package to which your Java programs belong.

  Execute the above command in the project folder of this lab,
  i.e. `CS2310\Labs\lucky_draw\`

  The command for running your Java application is:

  ```
  javac game/TUIRaffle
  ```

  where `TUIRaffle` is the class which contains the method `main`.

- Can you set up a raffle with the following 3 prizes?
  1. Family holiday at Disneyland Paris (3 nights), £1000
  2. Family holiday at Legoland Windsor (1 night), £250
  3. Tickets to Tumble Jungle (x4), £20
- The raffle simulator simulates 5 transactions for selling tickets. Each transaction is made with a single buyer, but the buyer can purchase $\geqslant 1$ ticket in a single transaction. Enter data for those 5 transactions.
- Observe the results of the raffle. Who has won a prize?

  (The more tickets a person buys, the higher the chances for him/her to win a prize.)

4 Further Challenges

1. Modify the application to include the handling of money obtained from ticket sales. Each ticket within the same raffle is sold at a fixed price. However, different raffles can have different prices for their tickets.
2. Draw an **UML class diagram** describing the relationship between **all** of the classes used in this application. In your diagram, you should include the standard Java API classes used in the application.
3. Rewrite method `main` in class `TUIRaffle` so that the ticket sale is not fixed at 5 transactions, but it will continue until the application is instructed otherwise.
4. Rewrite the software project without using the **for-each** (aka enhanced `for`) statement.

Hint: You will need to use the standard Java interface `java.util.Iterator` and its methods (i.e. `hasNext()` and `next()`) to accomplish the task.

➠ *What is the main advantage of using* **generic types** *in Java?*