

Lab 1. Defining Generic Types

Theme. In this practical, you will:

- practise the use of type variables
- practise defining 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): lab1.zip

1 Getting started...

1. In a web browser, download the archive `lab1.zip` from Blackboard and extract its contents into your default Eclipse workspace for this module.
2. Start up Eclipse.
3. Making use of the contents of your extracted archive, create a new Java project named `lucky_draw_generic` in your Eclipse workspace using the contents of your extracted archive.

2 Lab Tasks: Working with Generic Types and Type Variables

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. `TUIRaffle` does *not* require any run time argument.

- Can you set up a raffle with the following 3 prizes?
 1. Family holiday at Disneyland Paris (3 nights), £1500
 2. Family holiday at Legoland Windsor (1 night), £450
 3. Tickets to Tumble Jungle (x4), £25

- The raffle simulator simulates 5 transactions for selling tickets. Each transaction is made with a single buyer, but the buyer can purchase ≥ 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 them to win a prize.)

Hint:

- To run a Java application within eclipse:
 1. Select the top level class of this Java application (i.e. `.java`) within the Package Explorer tab.
 2. Select from the pull-down menu: `Run` \Rightarrow `Run Configurations...`
A new window titled “Run Configurations” will then pop up.
 3. On the left hand side of the “Run Configurations” window, there is a navigation menu with the item `Java Application`. If the name of the high level class (i.e. `TUIRaffle`) has not yet appeared as a sub-item underneath `Java Application`, double-click on the item `Java Application`. This will add the previously selected class to the list of runnable Java applications known by eclipse.
 4. Select the required runnable Java application (i.e. `TUIRaffle`).
 5. Press the button “Run”, when you are ready to run the Java application.
- 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 raffle/*.java
```

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

Execute the above command in the project folder of this lab.

The command for running your Java application is:

```
java raffle/TUIRaffle
```

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

4 Further Challenges

1. Describe the relationship between a generic class and type variable(s).
2. Briefly describe the main advantage of using **generic types** in Java.
3. Consider the following partial definition of the class `Auction`.
An `Auction` object keeps a list of lots modelled as `String` objects.

```
1 import java.util.ArrayList;
2 import java.util.Iterator;
3 /**
4  * A class to model simple behaviour of an auction
5  */
6 public class Auction
7 {
8     // The list of lots in this auction.
9     private ArrayList lots;
10
11     /**
12      * Constructor
13      * @param lotsItems a series of String objects modelling the lots in this Auction
14      */
15     public Auction(String... lotItems)
16     {
17         lots = new ArrayList();
18
19         for (String lot : lotItems) {
20             lots.add(lot);
21         }
22     }
23
24     /**
25      * Show the full list of lots in this auction.
26      */
27     public void showLots()
28     {
29         Iterator it = lots.iterator();
30
31         while(it.hasNext())
32         {
33             String lot = it.next();
34             System.out.println(lot.toString());
35         }
36     }
37
38     /**
39      * Show the full list of lots in this auction using a for-each loop.
40      */
41     public void showLots_withForEach()
42     {
43         for (String lot : lots) {
44             System.out.println(lot.toString());
45         }
46     }
47     // other details omitted
48 }
```

- (a) State the locations of the two syntax errors in class `Auction`.
What was the cause of each error?
 - (b) Write out the modification required to remove the errors.
4. Explain the relationship between a for-each statement and the Java interfaces `java.lang.Iterable` and `java.util.Iterator`.

5 Further Programming Exercises

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.