COMP503/ENSE502/ENSE602: Week 4 – Exercises

➢ **Exercise 01**

a) Design a **Book** class with instance variables storing the title of the book, the author's name and whether or not the book is currently borrowed. Write a **toString** method to return a String which, when printed to the console, is similar to the following output

```
The Lord of the Rings By J. R. R. Tolkien is borrowed: No
```

b) Design a **Library** class with the following attributes and methods:

- Contains a primitive array of **Book** objects.
- has a constructor with the method signature

public Library(int capacity)

which has an input parameter representing the initial capacity of the library (e.g. the length of the array). If the value supplied is less than 1, choose a default array length

- has a **toString** method which returns a formatted menu of the books stored in the library (see output below)
- has a method with the following return type and signature

public boolean addBook(Book book)

which puts the input book object into the first free location in the array and returns true. Otherwise, false is returned.

- has a method with the following return type and signature

public Book borrow(String title)

which takes as input a **String** containing the book's title. If the book is found in the library, it's status is set to borrowed and the Book object is returned. Otherwise, null is returned. (Hint: Use the **string1.equals(string2)** method to compare strings)

c) Write and execute the following **LibraryApp** class.

```
public class LibraryApp
{
	public static void main(String[] args)
	{
		Library library = new Library(5);
		library.addBook(new Book("The Lord of the Rings", "J. R. R.
			Tolkien"));
		library.addBook(new Book("Harry Potter and the Philosopher's
			Stone", "J. K. Rowling"));
		library.addBook(new Book("1984", "George Orwell"));
		library.addBook(new Book("Where the Wild Things Are", "Maurice
			Sendak"));
		library.addBook(new Book("The Hitchhiker's Guide to the
			Galaxy", "Douglas Adams"));
		System.out.println(library);

		Book aBook = library.borrow("1984");
		System.out.println("Book borrowed: " + aBook);
	}
}
```

Sample output of the **`LibraryApp`** class

Example output:

```
Contents of the library
1. The Lord of the Rings By J. R. R. Tolkien is borrowed: No
2. Harry Potter and the Philosopher's Stone By J. K. Rowling is borrowed: No
3. 1984 By George Orwell is borrowed: No
4. Where the Wild Things Are By Maurice Sendak is borrowed: No
5. The Hitchhiker's Guide to the Galaxy By Douglas Adams is borrowed: No

Book borrowed: 1984 By George Orwell is borrowed: Yes
```

COMP503/ENSE502/ENSE602: Week 4 – Exercises

➢ **Exercise 02**

Design the **Purchase** class which maintains an instance variable describing a purchase and it's price (e.g.: Book, $9.99). Your class should be fully encapsulated with useful constructor and toString methods.
Next, design the **Basket** class which will store your purchases. The Basket class will maintain an array of Purchase objects and a number *nPurchases* indicating how many purchases have been made so far. The class should have the following methods

- `Basket(int n)`(**NOTE**: this is constructor!), initializes the purchases array to a fixed length, specified by the input value. If the input is zero or negative, set the length to 10. Initialise the nPurchases instance variable.
- `public void addPurchase(Purchase p)` , adds a purchase to the array and increments the number of purchases stored. If there are already too many purchases, then the purchase cannot be added
- `public int getNPurchases()` returns the number of purchases
- `public Purchase getMostExpensive()` returns the most expensive purchase
- `public Purchase[] getPurchases()` , returns an array of Purchase objects. The returned array should not contain any null entries; e.g. it is of length nPurchases.
- `public double total()`, returns the total price of all purchases in the basket.
- `public void printReceipt()`, prints an itemised transcript of each purchase in the basket, it's price and the total price of all items. (Hint: use formatting for prices.)

Write a main method in another class called **BasketApp** which demonstrates the functionality of each of your methods.

COMP503/ENSE502/ENSE602: Week 4 – Exercises

➢ Exercise 03

Design the **NumberList** class which maintains a list of integers. Fix the length of the array to a constant value, say 10.
Write the following methods:

- A constructor which initializes the list to zero values.
- `public int size()` returns the length of the array
- `public String toString()` which prints the list of numbers.
- `private boolean valid(int i)` which returns true if an input index is in the range of the list and false otherwise
- `public void update(int index,int value)` updates the ith number of the array to value, assuming i is a valid array index. Otherwise, The array is unmodified.
- `public int min()` and `public int max()` return the smallest/largest number in the array
- `public int nonZero()` returns the number of integers that are not zero in the array
- `public double average()` returns the average of the numbers in the array
- `public int getNumber(int i)` returns the number at the ith position
- `public void absolute()` replaces each number in the array with it's absolute value e.g. [-1,-4,0,5] to [1,4,0,5]
- `public void scale(int f)` scales each number in the array by a factor. E.g. scaling [-1,-4,0,5] by 2 yields [-2,-8,0,10]
- `public NumberList sub(int startIndex,int endIndex)` returns a new NumberList object containing a list of the elements between the indicies startIndex and endIndex (inclusive). For example invoking sub(2,4) on the list [-1,-4,0,5,34,8,0,0,0,0] yields the NumberList object with [0,5,34,0,0,0,0,0,0,0]. If startIndex or endIndex is an invalid index, then a newly initialized NumberList object is returned.

Write a main method in another class called **NumberListTest** which demonstrates the functionality of each of your methods in number list class. Initialize each number in number list using random class!

COMP503/ENSE502/ENSE602: Week 4 – Exercises

➢ Exercise 04

Create the Java class called **SimpleDate** and complete the class with attributes `int day; int month; int year;` using the private accessor modifier.
Write:

- get methods for each attribute
- writing set methods for the instance variables such that
  - **day** can only be set to a value between 1 and 31,
  - **month** can only be set to values 1 to 12
  - **year** is a value between 2000 and the current year.

  If these conditions are not met by the set method's input parameters, choose appropriate default value to set the instance variable to instead.
- a method `public void setDate(int day, int month, int year)` that sets the instance variables using the set methods you've written.
- use setDate method to write constructor with the method signature

```
SimpleDate(int day,int month, int year)
```

- Choose some appropriate initial values for the SimpleDate attributes and use these values to write the default constructor.
- Write a toString method returning a string of the form "DD/MM/YYYY". To construct this string, use String concatenation **(+)** to form the required output.

Create a program **SimpleDateApplication** with a main method that tests the functionality of each SimpleDate method. For example, consider the following code snippet:

```
SimpleDate d1 = new SimpleDate();
d1.setDay(22);
d1.setMonth(3);
d1.setYear(2019);
System.out.println(d1);
SimpleDate d2 = new SimpleDate(14,03,2019)
System.out.println(d2);
```