TP09

Exceptions

Remarks:

- ✓ All classes need to have validations
- ✓ All validation errors are in form of Exceptions
  (example: “Invalid date of birth” ➔ BirthDateException
  where BirthDateException is your custom exception class)

Exceptions

When executing Java code, different errors can occur:

- Coding errors made by the programmer,
- Errors due to wrong input, or
- Other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (or throw an error).

Example:

```
public class MyClass {
    public static void main(String[ ] args) {
        int[] myNumbers = {1, 2, 3};
        System.out.println(myNumbers[10]); // error!
    }
}
```

To handle exception we use try…catch syntax:

```
public class MyClass {
    public static void main(String[ ] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        }
    }
}
```

The finally statement lets you execute code, after try...catch, regardless of the result.

For example:

```
public class MyClass {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

To generate exception, we use keyword throw:

```
public class MyClass {
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You must be at
least 18 years old.");
        }
        else {
            System.out.println("Access granted - You are old enough!");
        }
    }

    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }
}
```

TP09.1. Student class

Implement the "**Student**" class. It's necessary to save in the attributes the following data: *name, date of birth, telephone number, city, country, the group (A, B, …)*. Do the following tasks:

1. Implement the methods for a data input (read from keyboard and fill in the fields), for a data output,
2. implement the accessors (ex: getName() is accessor of name field) to access the separate attributes,
3. Implement the mutators (ex: setName(String name) is mutators of name field).

TP09.2. New year gift shop (cashier mode)

In TP6, we have created a class name **Product** that represents products to sell in the gift shop.

And then, we also wrote a program in Java to help a shop to manage the products to sell to customers. The program provided a menu for admin to:

1. List all products in shop with product number, name, price, and amount in stock
2. Add new product to the list
3. Remove product from list by index
4. Update product in list

Now, add Cashier menu:

1. Serve a customer
    a. List all products with price and amount available to sell, and cashier choose a product that customer want to buy and enable customer to pick many products and amount
    b. Calculate total price and display invoice
    c. Cashier can input discount as needed
    d. Can serve multiple customers.
2. Close today shop (end of serving customer and show total revenue for today)

TP09.3. Book class

Create a Java class represents **Book** information in GIC library. Class contains all needed information for librarian and students to:

1. Search (by title, description, category (math, Java, etc.), isbn, author(s), published date…)
2. Check availability for borrow
3. Check number of books in inventory
4. Decrease and increase number or books in inventory
5. Borrow (some books can be borrowed, some cannot)

The class Book need methods:

1. Data input (user input from keyboard to create new book)
2. Display book information

Note: This class should prevent all type of errors of input by user.

TP09.4. Book Category class

Using previously created class **Book** as references and reuse it. Create new class name **Category** represents category of the book. The category class should have:

1. Name, Description, ArrayList of Books
2. Data input to create new category
3. Data input to add/remove book(s) in the category
4. Method to list all books
5. Find books in this category (by ISBN or Title)
6. Count books in this category

TP09.5. Library class

Using previously created classes above as references and reuse them. Create new class name **Library** represents GIC library. This class should be able to:

1. List all categories
2. List books by categories
3. List books by year
4. List available books
5. Add new books
6. Decrease book by isbn
7. Remove book by isbn
8. Add new copies of book
9. Mark book not available to borrow
10. Let student borrow the book, max 5 books, max 1 week
    (if the student not yet return the previous borrowed books, he/she can't borrow more)
11. List students that borrowed the books
12. Let student to return book