Lab 8 - Functional Interface and Lambda Expressions

This lab contains in-class exercises related to functional interface and lambda expressions

Task 1: The `RunnableExample` shows how to build threads in two ways:

a. **Using `Runnable` object**
b. **Using anonymous `Runnable` object**
c. **Can you give an example of variable capture?**

Show how to build a thread with with lambda expression:

```
public class RunnableExample {

    public static void main(String[] args) {

        /** example 1
         * thread built with a Runnable object r
         */
        Runnable r = new Runnable() {
            public void run() {
                for (int i = 0; i < 2; i++)
                    System.out.println("Thread from Runnable r object");
            }
        };
        new Thread(r).start();

        /** example 2
         * thread built with anonymous Runnable object
         */
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 2; i++)
                    System.out.println("Thread from anonymous Runnable");
            }
        }).start();

        /** example 3
         * thread built with with lambda expression
         */
        new Thread ???
    }
}
```

Task 2: The class `Learner` and the functional interface `Exam` are used in the class `ExamResults`. In the class there is a method called

`public static String result(Learner learner, Exam exam)`

That returns the result of the exam for the learner. The method does not have an implementation. You are asked to complete the implementation of this method and to invoke it in the main.

```
@FunctionalInterface
interface Exam {
     String getExamResult(Learner learner);
}

//simple class to define a Learner
class Learner {

    public String name;

    public Learner(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Learner{" + "name='" + name + "}";
    }
}

public class ExamResults {

    /**
     * It returns the result of the exam result for a learner
     * @param learner lerner object
     * @param exam exam object
     * @return a string as the result of the exam
     */
    public static String result(Learner learner, Exam exam) {
        /* ??? */
    }

    public static void main(String[] args) {

        Learner l = new Learner("John");
        //the result method takes two arguments a learner and an exam
        String s = ExamResults.result(/* ???*/);
        System.out.println(l + " result " + s);
    }
}
```

Task 3: The class `PredicateExercise` is defined to filter a collection of students.
You are asked to select all students that are male, whose names contains the character 'e' and have gpa > 3.5

Fill out the missing part, compile and run your program.

Given
```
Student[] sa = {new Student("John", 'M', 3.1f),
                new Student("Wei", 'M', 3.9f),
                new Student("Lo", 'F', 3.8f),
                new Student("Peter", 'M', 3.8f),
        };
```

Your program must print:

[Student{name='Wei', gender=M, gpa=3.9}, Student{name='Peter', gender=M, gpa=3.8}]

```
class Student {

    private String name;
    private Character gender;
    private Float gpa;

    public Student(String name, Character gender, Float gpa) {
        this.name = name;
        this.gender = gender;
        this.gpa = gpa;
    }

    public String getName() {
        return name;
    }

    public Character getGender() {
        return gender;
    }

    public Float getGpa() {
        return gpa;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", gender=" + gender +
                ", gpa=" + gpa +
                '}';
    }
}
```

```
public class PredicateExercise {

    List<Student> students = new ArrayList();

    public PredicateExercise(List<Student> students) {
        this.students = students;
    }

    /**
     * Define the predicate conditions
     *
     * @return lambda expression Predicate
     */

    public Predicate<Student> findStudent() {
        return /* ??? */
    }

    /**
     * Filter a list of objects of type Student
     * (it uses Stream that will be explained later in the course)
     *
     * @param ls        list of students
     * @param predicate lambda expression
     * @return list of students filtered use the predicate
     */

    public List<Student> filterStudents(List<Student> ls,
                                        Predicate<Student> predicate) {

      return
      students.stream().filter(predicate).collect(Collectors.<Student>toList())
    }

    public static void main(String[] args) {
        Student[] sa = {new Student("John", 'M', 3.1f),
                new Student("Wei", 'M', 3.9f),
                new Student("Lo", 'F', 3.8f),
                new Student("Peter", 'M', 3.8f),
        };

        PredicateExercise pe = new PredicateExercise(Arrays.asList(sa));

        System.out.println(/* ??? */);
    }
```