Abdulrasheed Mustapha

Coding Score: 813

BASIC INFO

Email: abdulrasheedmustapha66@gmail.com

Test: [JPMC] Core Java Test

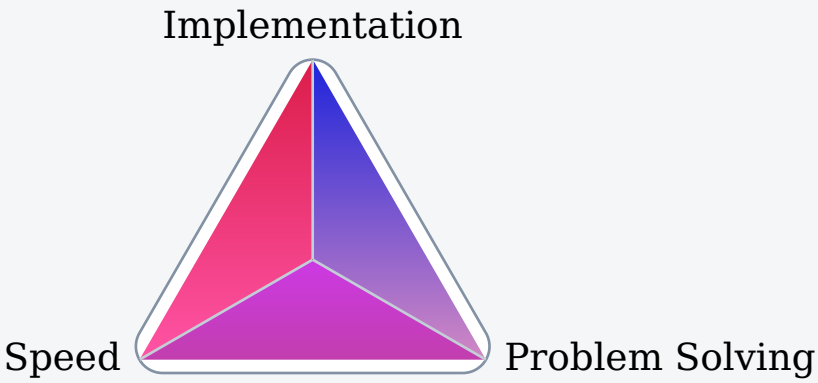
Finished On: 17 Apr 2020

Solved: 8/10

Score: 840/1000

Time Taken: 27m/45m

CORE SKILLS



Labels: -

Task	Solve Time	Score	Similarity
addKbeforeFs	6min	300/300	none
javaAbstractClassInstance	1min	100/100	-
javaPrintPerson1	1min	100/100	-
javaArrayVector	4min	60/60	-
javaFinally	6min	0/100	-
javaThreads	1min	60/60	-

javaOopMethodOverriding	12sec	100/100	-
javaMethodVariableInMemory	2min	60/60	-
javaOopPrinciples	8sec	60/60	-
javaThreadStack	4min	0/60	-

Task details: **addKbeforeFs**

Description:

You are given a string `text` consisting of English letters and spaces. Find all the capital and lowercase `F`s in it and insert a capital `K` right before each one of them.

Solution (main.java):

```
String addKbeforeFs(String text) {
    String newText="";

                for(int i =0; i<text.length(); i++) {
                        if(text.charAt(i) == 'f' || text.charAt(i) == 'F') {
                                newText += "K";
                                newText += text.charAt(i);
                        }
                        else{
                                newText += text.charAt(i);
                        }
                }

                return newText;
}
```

Task details: **javaAbstractClassInstance**

Description:

Given the following class

```
abstract class Animal {
    abstract void sayHello();
}
```

What will the following code do?

```
Animal animal = new Animal();
animal.sayHello();
```

- ○ Nothing. *(Incorrect)*

- ✓ Code will not be executed - compilation error will occur. It is not possible to create an instance of class *Animal*. *(Correct)*

- ○ Code will not be executed - compilation error will occur. Class *Animal* cannot have methods without implementation. *(Incorrect)*

- ○ Code will be executed, but will throw an exception. *(Incorrect)*

Task details: **javaPrintPerson1**

Description:

What needs to be done to see string “person1” as an output?

```
import java.util.HashMap;

public class Person {
    private String surname;
    private String name;

    public Person(String surname, String name) {
        this.name = name;
        this.surname = surname;
    }

    public static void main(String[] args) {
        HashMap<Person, String> personMap = new HashMap<>();
        Person person1 = new Person("Smith", "John");
        Person person2 = new Person("Smith", "John");

        personMap.put(person1, "person1");
        System.out.println(personMap.get(person2));
    }
}
```

- ○ Inherit class *Person* from class *HashMap.Entry*. *(Incorrect)*
- ● Implement *equals* and *hashCode* methods in class *Person*. *(Correct)*
- ○ Implement copy constructor in class *Person*. *(Incorrect)*
- ○ Do nothing. *(Incorrect)*

Task details: **javaArrayVector**

Description:

What is the difference between *Array* and *Vector* classes?

- ○ *Vector* can store object references whereas *Array* can't.

  (Incorrect)

- ✓ *Vector* is thread-safe whereas *Array* is not.

  (Correct)

- ○ *Vector* can be returned from a method whereas *Array* can't.

  (Incorrect)

- ○ *Array* elements can be changed, but not *Vector*'s

  (Incorrect)

Task details: **javaFinally**

Description:

What kind of errors does the following code contain?

```
final class Parent {
    protected int age;

    @Override
    void finalize() throws Throwable {
        System.out.println("finalize() was executed in Parent class");
    }
}

class Child extends Parent {
    @Override
    public void finalize() throws Throwable {
        System.out.println("finalize() was executed in Child class");
    }

    int compare(Parent parent) {
        try {
            if (age >= parent.age) {
                throw new IllegalStateException("Child can't be older than parent");
            }
            return age - parent.age;
        } finally {
            System.out.println("Child age: " + age);
        }
    }

    final int gender() {
        return 0;
    }
}

class Son extends Child {
    @Override
    final int gender() {
        return -1;
    }
}

class Daughter extends Child {
    @Override
    int gender() {
        return 1;
    }
}
```

- ☐ *Child* class **cannot** be inherited from the *Parent* class.

(Correct)

- [x] *Son* class **cannot** be inherited from the *Child* class.

(Correct)

- [x] *Daughter* class **cannot** be inherited from the *Child* class.

(Correct)

- [x] Method `compare` in the *Child* class **should contain** `throws` keyword in its definition.

(Incorrect)

- [x] The *Parent* class **cannot** inherit `finalize` method and override it.

(Correct)

- [ ] The *Child* class **cannot** inherit `finalize` method and override it.

(Incorrect)

- [ ] None of the above *(Incorrect)*

Task details: **javaThreads**

Description:

How can you create an instance of a *Thread* class in Java?

A. Extend *Thread* class
B. Implement *Callable* interface
C. Implement *Runnable* interface
D. Implement *Startable* interface
E. Implement *Executable* interface

Which of the items listed above are true?

- ○ B, E. *(Incorrect)*
- ○ C, D, E. *(Incorrect)*
- ✓ A, C. *(Correct)*
- ○ A, B, C. *(Incorrect)*

Task details: **javaOopMethodOverriding**

Description:

What principle is used in the following code?

```
public class Animal {
    public void move() {
        System.out.println("Animal can move");
    }
}

class Cat extends Animal {
    @Override
    public void move() {
        System.out.println("Cat can walk and run");
    }
}
```

- ○ Encapsulation. *(Incorrect)*
- ○ Manipulation. *(Incorrect)*
- ○ Dependency Inversion. *(Incorrect)*
- ✓ Inheritance. *(Correct)*

Task details: **javaMethodVariableInMemory**

Description:

```
public class Test {
    public static void main(String[] args) {
        int variableInMemory = 42;
        System.out.println("Variable in memory: " + variableInMemory);
    }
}
```

In which section of JVM memory `variableInMemory` would reside?

- ○ On the Queue. *(Incorrect)*
- ○ On the Deque. *(Incorrect)*
- ✓ On the Stack. *(Correct)*
- ○ On the Heap. *(Incorrect)*
- ○ Neither on the Heap nor on the Stack, but in Method Area. *(Incorrect)*

Task details: **javaOopPrinciples**

Description:

What are the major four principles of object oriented programming?

- ○ Inheritance, Polymorphism, Encapsulation, Aggregation.

  (Incorrect)

- ✓ Inheritance, Polymorphism, Encapsulation, Abstraction.

  (Correct)

- ○ Composition, Abstraction, Manipulation, Dependency Injection.

  (Incorrect)

- ○ Single Responsibility, Interface Segregation, Dependency Inversion, Liskov Substitution.

  (Incorrect)

Task details: **javaThreadStack**

Description:

What is the relationship between threads and stacks?

- ○ Each thread has its own single stack, and each stack is owned by one thread.

  (Correct)

- ✓ Each thread can have any number of stacks, and each stack is owned by one thread.

  (Incorrect)

- ○ Each thread can have any number of stacks, and each stack can be owned by multiple threads.

  (Incorrect)

- ○ Each thread can has its own single stack, and each stack can be owned by multiple threads.

  (Incorrect)

- ○ All threads share a single stack.

  (Incorrect)