Question **1**

Correct

Mark 100.00 out of 100.00

Time limit	1 s
Memory limit	64 MB

DND

Dungeon and Dragon atau biasa disingkat DnD adalah sebuah permainan peran (role-playing). DnD dimainkan dengan cara bercerita (story telling) yang melibatkan pemain untuk membuat karakter dalam dunia imaginatif dan melakukan berbagai aksi didalamnya. Anda diminta untuk membuat simulasi DnD sederhana. Untuk membantu implementasi, anda akan diwajibkan menggunakan salah satu design pattern yaitu **Strategy**. Nantinya sebuah karakter akan mempunyai kelas tertentu dengan kemampuan tertentu. Karakter dapat memilih kelas tertentu sesuai dengan gaya bermain masing-masing.

Strategy memungkinkan penggunaan algoritma yang berbeda pada suatu objek atau kelas yang diatur pada waktu runtime, disesuaikan dengan konteks atau kebutuhan yang dihadapi. Dengan **Strategy**, kita dapat memisahkan algoritma kemampuan dari tiap kelas tanpa mempengaruhi implementasi dari kode karakter.

Disediakan DnD.zip yang berisikan

1. Main_Dummy.java **<== Pendukung pengujian**
2. Kelas.java
3. Karakter.java
4. Domba.java **<== Perlu diimplementasikan**
5. Pendekar.java **<== Perlu diimplementasikan**
6. Pembunuh.java **<== Perlu diimplementasikan**
7. Penyihir.java **<== Perlu diimplementasikan**

Kumpulkan kembali **semua file yang perlu diimplementasikan saja** dalam zip dengan nama dan format sama yaitu **DnD.zip**. Disediakan Main_Dummy.java untuk membantu melakukan pengujian.

Lebih lanjut terkait Strategy Pattern cek disini

Java 8

DnD.zip

Score: 80

Blackbox

Score: 80

Verdict: Accepted

Evaluator: Exact

No	Score	Verdict	Description
1	10	Accepted	0.26 sec, 26.92 MB
2	10	Accepted	0.26 sec, 28.07 MB
3	10	Accepted	0.34 sec, 27.93 MB
4	10	Accepted	0.21 sec, 29.96 MB
5	10	Accepted	0.15 sec, 28.00 MB
6	10	Accepted	0.16 sec, 27.93 MB
7	10	Accepted	0.37 sec, 28.18 MB
8	10	Accepted	0.13 sec, 28.98 MB

Question **2**

Correct

Mark 100.00 out of 100.00

Time limit	1 s
Memory limit	64 MB

Buatlah sebuah program untuk melakukan validasi email. Email dianggap valid apabila memenuhi aturan berikut

- Email harus mengandung tepat 1 karakter @ ditengah email
- Bagian sebelum @ tidak boleh kosong
- Bagian email setelah @ harus memiliki tepat 1 buah titik (.)
- Email hanya akan mengandung huruf, angka, (@), dan (.). Karakter lain tidak perlu diperiksa
- "john@example.com" => Email Valid
- "jane.doe@gmail.com" => Email Valid

Sebagai contoh email tidak valid apabila:

- "" => mengembalikan pesan "Email tidak boleh kosong"
- "example.com" => mengembalikan pesan "Email harus mengandung tepat satu buah @"
- "@example.com" => mengembalikan pesan "@ tidak boleh di awal email"

Domain email tidak valid apabila:

- "john@com" => mengembalikan pesan "Email harus memiliki domain yang valid"
- "john@.com" => mengembalikan pesan "Email harus memiliki domain yang valid"
- "john@com." => mengembalikan pesan "Email harus memiliki domain yang valid"

Lengkapi dan submit file `Email.java`.

File tersebut akan memiliki 3 buah kelas, yaitu `Email`, `InvalidEmailException` yang mengextend kelas Exception, dan `InvalidDomainException` yang mengextend kelas Exception. `InvalidEmailException` akan menerima input berupa pesan custom ketika diinisiasi. Sedangkan, `InvalidDomainException` akan mengembalikan string mutlak ketika domain email tidak valid.

Dilarang menambahkan fungsi baru kedalam file `Email.java`

Java 8

Email.java

Score: 100

Blackbox

Score: 100

Verdict: Accepted

Evaluator: Exact

No	Score	Verdict	Description
1	12	Accepted	0.08 sec, 28.65 MB
2	12	Accepted	0.07 sec, 28.08 MB
3	12	Accepted	0.08 sec, 28.83 MB
4	12	Accepted	0.07 sec, 28.36 MB
5	12	Accepted	0.08 sec, 27.86 MB
6	12	Accepted	0.07 sec, 28.00 MB
7	12	Accepted	0.08 sec, 29.93 MB
8	16	Accepted	0.08 sec, 27.79 MB

Question **3**

Requires grading

Marked out of 100.00

Time limit	1 s
Memory limit	64 MB

Design pattern merupakan best practice dalam menyelesaikan sebuah permasalahan di dunia pemrograman. Salah satu design pattern yang ada merupakan **template method**. Secara singkat, design pattern ini memecah urutan eksekusi menjadi method-method dan terdapat salah satu method besar yang menggabungkan method-method pecahan sebelumnya menjadi urutan eksekusi yang lengkap. Lebih lengkapnya: referensi.

Di salah satu section Tubes 2, kalian disarankan untuk menggunakan design pattern **adapter**, namun alternatif lain adalah menggunakan design pattern **template method**. Agar memahami bagaimana implementasinya, silahkan implementasi kelas-kelas di bawah ini dengan ketentuan berikut.

- Diberikan asbtract class Processor.java.
- Implementasikan 3 class yang meng-extend Processor.java, yaitu ImplementorA.java, ImplementorB.java, dan ImplementorC.java.
- Implementasi setiap class sedemikian rupa sehingga ketika method check() dan run() dipanggil menghasilkan output sebagai berikut:

*Output merupakan text yang dikeluarkan menggunakan fungsi System.out.println().

Input (Potongan Kode)	Output	Keterangan
	Implementor A running process 1.	
Processor pA = new ImplementorA()	Implementor A running process 2.	
pA.check()	Implementor A running process 3.	-
pA.run()	Implementor A running process 1.	
	Implementor B running process 2.	
Processor pB = new ImplementorB()	Implementor B running process 3.	
pB.check()	Implementor B running process 4.	-
pB.run()	Implementor B running process 3.	
	Implementor C running process 3.	
Processor pC = new ImplementorC()	Implementor C running process 4.	
pC.check()	Implementor C running process 5.	
pC.run()	Implementor C running process 5.	

Java 8

TemplateProcessor.zip

Score: 60

Blackbox

Score: 60

Verdict: Accepted

Evaluator: Exact

No	Score	Verdict	Description
1	10	Accepted	0.53 sec, 27.83 MB
2	10	Accepted	0.69 sec, 28.06 MB
3	10	Accepted	0.59 sec, 27.96 MB
4	10	Accepted	0.44 sec, 29.39 MB
5	10	Accepted	0.29 sec, 28.24 MB

No	Score	Verdict	Description
6	10	Accepted	0.75 sec, 30.09 MB

Question **4**

Tries remaining: 10

Marked out of 100.00

Time limit	1 s
Memory limit	64 MB

Terdapat sebuah class Reflection.java dengan isi sebagai berikut

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Reflection{
    //TIDAK BOLEH MENGUBAH NAMA METHOD YANG SUDAH ADA DAN PARAMS SERTA INPUT TYPENYA
    //BOLEH MENAMBAHKAN PRIVATE / PROTECTED METHOD SESUAI DENGAN KEBUTUHAN
    //DI LUAR SANA ADA KELAS YANG NAMANYA Ghost DAN Secret.

    public ArrayList<String> ghostMethods(){

        return new ArrayList<>();
    }

    public Integer sumGhost() throws Exception{

        return 0;
    }

    public String letterGhost() throws Exception{

        return "";
    }

    public String findSecretId(List<Secret> sl, String email) throws Exception{

        return "NaN";
    }
}
```

Tugas anda adalah implementasi ke empat method tersebut dengan ketentuan sebagai berikut

1. Terdapat sebuah kelas `Ghost` yang memiliki N method, dan semua method tersebut adalah private. Sesuai namanya, yaitu `Ghost`, maka isi kelas tersebut adalah misterius dan tidak diketahui baik nama method maupun return dari method tersebut (dipastikan return dari Ghost antara Integer dan String saja).
2. Implementasi method `getMethod()` dimana akan membaca seluruh method yang dimiliki oleh kelas `Ghost` dan menyimpan nama method tersebut ke dalam ArrayList
3. Implementasi method `sumGhost()` dimana akan menjumlahkan seluruh return dari method dari kelas `Ghost` yang memiliki kembalian Integer
4. Implementasi method `letterGhost()` dimana akan mengembalikan concat dari seluruh method di kelas `Ghost` yang memiliki kembalian String. (Ketika concat langsung concat aja, gausah di kasih spasi atau pemisah lainnya).
5. Terdapat kelas lain yang bernama `Secret` dengan isi sebagai berikut

```
public class Secret{
    private String email;
    private String nama;
    public static Integer counter = 0;
    private String uniqueId;

    //KELAS SUDAH PATEN, TIDAK BOLEH DIEDIT EDIT

    public Secret(String email, String nama){
        //MAGIC HAPPENED THERE.
        //Intinya konstruktor biasa, cuma ada randomisasi buat uniqueId, dan jangan lupa
        //counter++ pada bagan akhir konstruktor ini.
    }

    public boolean isThis(String email){
        //GUNAKAN FUNGSI INI UNTUK MENCOCOKAN EMAIL DENGAN EMAIL YANG DIMILIKI KELAS
        return this.email.equalsIgnoreCase(email);
    }

}
```

1. Implementasi method `findSecretId(List<Secret> sl, String email)` yang berfungsi akan mencari `Secret` sesuai dengan email yang hendak dicari. Apabila `Secret` dengan email yang dimaksud ada, maka kembalikan `uniqueId` dari email tersebut, jika tidak

kembalikan `NaN` sesuai dengan template yang ada diatas.

2. Tips, gunakan `setAccessible(true)` untuk mengakses method maupun attribut private / protected.
3. Untuk implementasi no. 6, manfaatkan method `isThis(String email)`
4. Untuk debugging di local teman-teman bisa membuat kelas `Secret` sesuai dengan template diatas, lalu memberikan uniqueId dengan String random, dan kelas `Ghost` dengan membuat beberapa method private yang mengembalikan Integer dan String.

kumpulkan `Reflection.java`

Java 8

Maximum size for new files: 512MB, maximum attachments: 1

Files

You can drag and drop files here to add them.

Run Check

Question **5**

Tries remaining: 10

Marked out of 100.00

Time limit	1 s
Memory limit	64 MB

Long running task merupakan sebuah perintah eksekusi yang memakan waktu lama. Jika perintah eksekusi ini dilakukan di *main thread* program, maka akan mengakibatkan program utama *hang* selama proses eksekusi berjalan. Oleh karena itu dibutuhkan pemanfaatan **multithreading** agar *long running task* tetap dapat dieksekusi namun tidak membuat program utama *hang*.

Diberikan Main.java, Client.java, Task.java, dan Worker.java untuk menjadi dasar pengetahuan kalian bagaimana **multithreading** bekerja di Java. Sayangnya implementasi Worker masih kurang efisien (gimana sih asistennya>:( ), perhatikan bahwa untuk setiap task diperlukan worker baru untuk menjalankan task tersebut. Tugas kalian adalah menunjukkan bahwa kalian lebih OP dari asisten dengan cara mengimplementasikan WorkerV2.java.

*Jangan ubah isi dari class-class yang telah ada kecuali WorkerV2.java dan Main.java (untuk keperluan test)
**File yang dikumpul hanya WorkerV2.java

Java 8

Maximum size for new files: 512MB, maximum attachments: 1

Files

You can drag and drop files here to add them.

Run Check

Question **6**

Tries remaining: 10

Marked out of 100.00

Time limit	1 s
Memory limit	64 MB

Pada soal ini, kita akan mencoba mendapatkan data-data yang dimiliki setiap mahasiswa ITB. Setiap Mahasiswa pasti memiliki NIM, IPK dan daftar matkul yang pernah diikuti.. Berikut adalah garis besar kelas Mahasiswa yang dimaksud.

```
import java.util.*;



class Mahasiswa {

  private int NIM;

  private double IPK;

  private List matkul;



  Mahasiswa(int NIM, double IPK) {

    this.NIM = NIM;

    this.IPK = IPK;

    this.matkul = new ArrayList();

  }



  private void addMatkul(String matkul) {

    this.matkul.add(matkul);

  }

}
```

Sebagai mahasiswa yang iseng, kita ingin melihat NIM, IPK, dan menambahkan matkul-matkul susah ke daftar matkul teman kita. Sayangnya atribut NIM, IPK dan method untuk menambahkan matkul bersifat private di kelas Mahasiswa, sehingga kita harus menggunakan reflection untuk mengaksesnya.

Buatlah kelas MahasiswaDecoder yang memiliki interface sebagai berikut

```
import java.lang.reflect.*;

public class MahasiswaDecoder {



  MahasiswaDecoder(Mahasiswa mahasiswa) {

  }



  public void addMatkul(String name) throws Exception {

  }



  public int getNIM() throws Exception {

  }



  public double getIPK() throws Exception {

  }



}

}
```

Hint: perhatikan kalau Field/Method private, bagaimana cara mengaksesnya? (gunakan method `setAccessible`)

Kumpulkan **MahasiswaDecoder.java**

Java 8

Maximum size for new files: 512MB, maximum attachments: 1

Files

You can drag and drop files here to add them.

Run Check

◄ Latihan Praktikum 5

Jump to...

Slide Responsi 6 ►