serrano
Mark: 8.9/20 (total score: 13.3/30)

+9/1/56+

TEST

QCM

Introduction à la programmation orientée objet
30/09/2016

Nom et prénom : SERRANO Simon

Groupe : 3

Vous apporterez un très grand soin à la présentation car elle interviendra dans la notation. Par exemple, les réponses très peu lisibles ou contenant du code non indenté ***seront considérées comme fausses.*** *Par ailleurs, la qualité du code proposé et la complexité des solutions interviendront dans la notation. Les questions à choix multiples marquées d' un trèfle admettent plusieurs réponses possibles alors que les autres questions n'en admettent qu'une. Les bonnes réponses apportent des points positifs,* ***les mauvaises réponses peuvent apporter des points négatifs.***

Question 1 ♣ Quelles affirmations s'appliquent aux tableaux :

1.6/2

- [x] sont indexés exclusivement par des entiers non-négatifs
- [ ] sont créés par, eg, `p = new Person(14)`
- [x] l'ordre de stockage des éléments est bien défini
- [x] sont de taille fixe
- [ ] sont dans le package `java.util`
- [x] peuvent stocker des primitifs, eg, `double`
- [ ] le nombre d'éléments est donné par `.size()`
- [x] sont déclarés par, eg, `Person[] p`

Question 2 ♣ Quelles affirmations s'appliquent aux listes :

1.75/2

- [ ] sont déclarés par, eg, `Person[]`
- [x] peuvent stocker des primitifs, eg, `double`
- [x] sont indexés exclusivement par des entiers non-négatifs
- [ ] sont créés par, eg, `new Person(14)`
- [x] l'ordre de stockage des éléments est bien défini
- [x] le nombre d'éléments est donné par `.size()`
- [ ] sont de taille fixe
- [x] sont dans le package `java.util`

Question 3 ♣ Quelles affirmations s'appliquent aux maps :

1.5/2

- [x] l'ordre de stockage des éléments est bien défini
- [x] le nombre d'éléments est donné par `.size()`
- [x] sont indexés exclusivement par des entiers non-négatifs
- [x] peuvent stocker des primitifs, eg, `double`
- [ ] sont initialsés par, eg, `new Person(14)`
- [x] sont dans le package `java.util`
- [ ] sont déclarés par, eg, `Person[]`
- [ ] sont de taille fixe

Question 4 ♣ On a `public class Toto extends Fred`. Laquelle (lesquelles) des expressions est (sont) valable(s) :

2/2

- [x] `Fred f = new Fred();`
- [x] `Toto t = new Toto();`
- [ ] `Toto t = new Fred();`
- [x] `Fred f = new Toto();`

Question 5 Le code

```
public interface Fooable {
    private int counter;
}
```

est légal.

0/1 ☒ vrai ☐ faux

Question 6 Le code

```
public class Foo {
    private int counter;
}

public class Bar {
    public Bar(int counter) {
        super(counter);
    }
}
```

1/1 ☐ compile mais donne une erreur d'exécution ☐ compile et exécute correctement
☒ ne compile pas

Question 7 Quel code est le supérieur du point de vue de maintenabilité

0/1 ☒ Les deux sont strictement pareil

```
public class Foo {
    private int counter;

    public Foo() {
        this(4);
    }

    public Foo(int counter) {
        this.counter = counter;
    }
}
```

☐

```
public class Foo {
    private int counter;

    public Foo() {
        counter = 4;
    }

    public Foo(int counter) {
        this.counter = counter;
    }
}
```

Question 8 Le code

```
public class Foo {
}

public class Bar extends Foo {
}

public class Toto {
    private List<Foo> fool = new ArrayList<>();

    public Toto(Bar bar) {
        fool.add(bar);
    }
}
```

0/1 ☒ ne compile pas ☐ compile mais donne une erreur d'exécution
☒ compile et exécute correctement

+9/3/54+

Question 9 ♣ Quel problème(s) peut-on relever sur le code suivant:

```
public class Account {
  private String owner;
  int money;

  public boolean equals(Account other){
    if (this.owner == other.owner){
      return true;
    } else {
      return false;
  }
}
```

- [x] le test va renvoyer false si on compare une instance avec elle-même.
- [x] ce code risque de provoquer des NullPointerExceptions.
- [ ] la méthode equals ne peut être redéfinie.
- [x] ce code peut générer des exceptions si le type de other est incorrect.

0.5/2

Question 10 Soit le code ci-dessous:

```
public class Supper {
  public String go() {
    return "Supper#go"
  }

  public void main() {
    System.out.println(go);
  }
}

public class Sube extends Supper {
    @Override
    public String go() {
      return "Sube#go";
    }
}
```

Quel est le résultat de l'exécution du code ci-dessous ?

```
public class Sube extends Supper
  public static void main(String[] args) {
    Supper s = new Sube();
    s.main();
  }
}
```

☐0 ☐1 ☐2 ☒3

1/1

Sube #go

Question 11 Expliquez pourquoi (question précédente). ☐0 ☐1 ☒2 ☐3

On crée un objet s de la classe Sopper en faisant appel au constructeur de la classe Sube.

En appelant la méthode main() de la classe Sopper, on appel la méthode go().

La méthode go() qui est appelée est finalement celle de la classe Sube, car l'objet a été créé avec le constructeur de cette classe.

Sube s = new Sube(...) ?

Question 12 Le code suivant définit deux classes. Quelle est la meilleure solution pour améliorer ce code ?

```
class Plane {
  private int speed, locationx, locationy;
  public void setSpeed(int speed) {if (speed>90) this.speed=speed;}
  public void setLocation(int locationx, int locationy){
    this.locationx=locationx; this.locationy=locationy;
  }
}

class Balloon {
  private int speed, locationx, locationy;
  public void setSpeed(int speed) {this.speed=speed;}
  public void setLocation(int locationx, int locationy){
    this.locationx=locationx; this.locationy=locationy;
  }
}
```

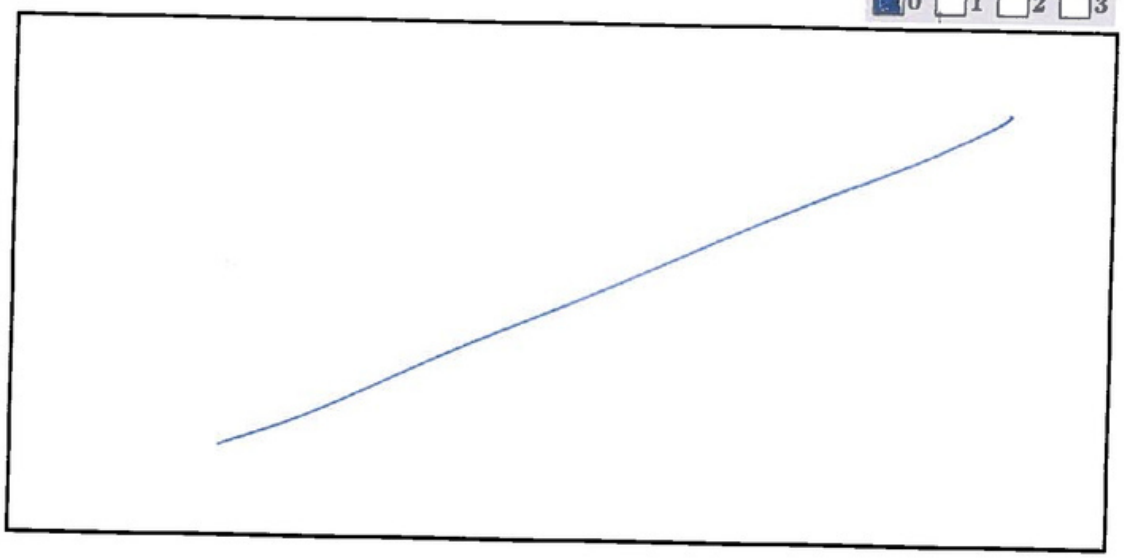
- [x] créer une classe abstraite VehiculeAerien contenant l'implémentation de setLocation et séci-fiant setSpeed
- [ ] créer une interface commune VehiculeAerien définissant setSpeed et setLocation.
- [ ] s'appuyer sur Class<Plane> et Class<Balloon> pour redéfinir le code de setSpeed().
- [ ] utiliser la méthode instanceOf() pour redéfinir le code de setSpeed().

0/1

Question 13 Présentez un exemple de code qui illustre le fonctionnement du typage dynamique.

0 1 2 3

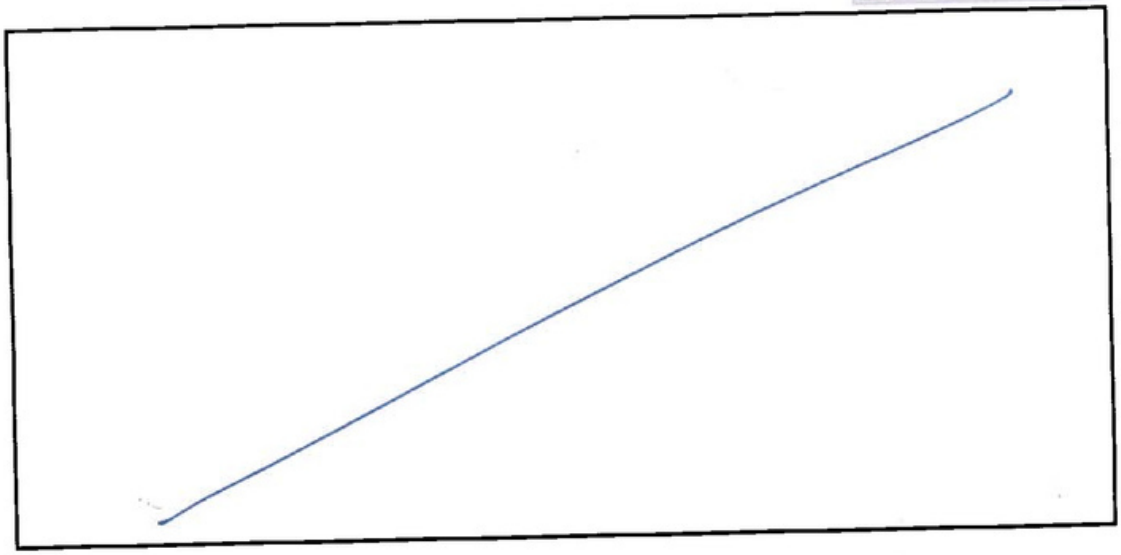
0/3



Question 14 Présentez un exemple de code qui illustre le fonctionnement du typage statique.

☒ 0 ☐ 1 ☐ 2 ☐ 3

0/3



Question 15 On cherche à modéliser des orchestres. Quel est le résultat du code suivant ?

```
public class ClassicalOrchestra {
  public String getTypeOfMusic() {return "Classical Music";}
}

public class ChamberOrchestra extends ClassicalOrchestra {
  public String getTypeOfMusic() {return "Chamber Music";}
}

public class SymphonicOrchestra extends ClassicalOrchestra {
  public String getTypeOfMusic() {return "Symphonic Music";}
}

class Main {
  public static void main(String[] args) {
    ClassicalOrchestra o = new ChamberOrchestra();
    System.out.println(o.getTypeOfMusic());
}
```

- ☐ "Symphonic Music"
- ☒ "Chamber Music"
- ☐ "Classical Music"
- ☐ NullPointerException

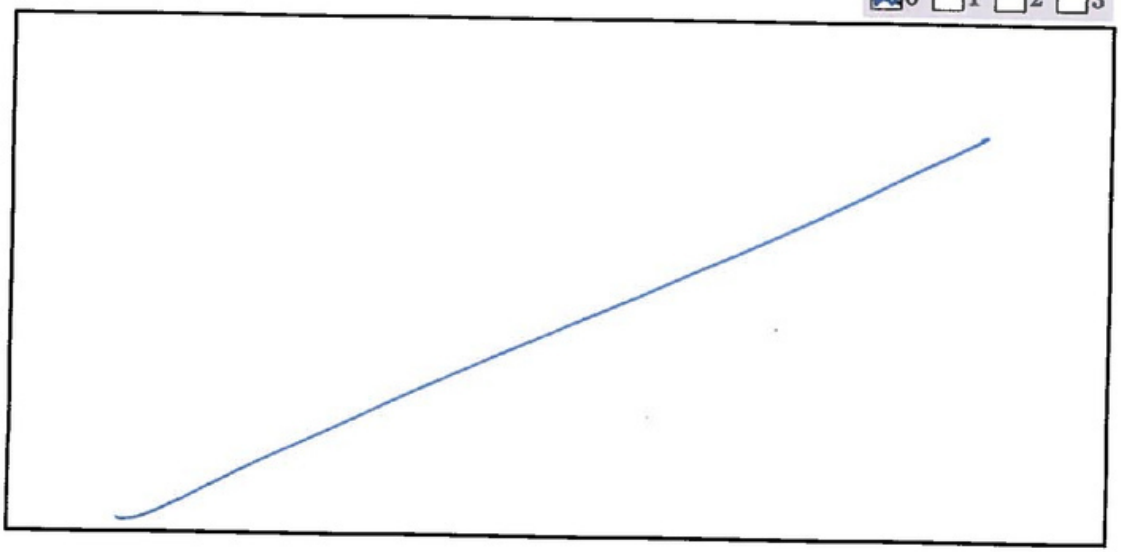
1/1

Question 16 Peut-on compiler et exécuter le code suivant ? Expliquez

```
class Main {
  public static void main(String[] args) {
    ClassicalOrchestra o = new ChamberOrchestra();
    SymphonicOrchestra so = (ChamberOrchestra) o;
    System.out.println(so.getTypeOfMusic());
    ChamberOrchestra co = (ChamberOrchestra) o;
    System.out.println(so.getTypeOfMusic());
    SymphonicOrchestra so = (SymphonicOrchestra) o;
    System.out.println(so.getTypeOfMusic());
}
}
```

☒ 0 ☐ 1 ☐ 2 ☐ 3

0/3



Question 17 ♣ Quelles sont les caractéristiques *recherchées* dans une application logicielle ?

0.5/0.5

- ☒ Low coupling
- ☒ High cohesion
- ☐ High coupling
- ☐ Low cohesion

Question 18 ♣ Quelles sont les caractéristiques *à éviter* dans une application logicielle ?

0.5/0.5

- ☒ Low cohesion
- ☒ High coupling
- ☐ High cohesion
- ☐ Low coupling