Examen 2014-2015 - Durée : 1h30

POO : Programmation Orientée Objet

Première année

Seule une feuille recto-verso d'aide mémoire manuscrite est autorisée.

Nom :	Prénom :

▷ **Question 1.** Lequels des exemples ci-dessous définissent de manière correcte une classe Java ? (2 réponses)

☐
```
#define TAB_SIZE;
class EvilCode {
  int[] t = new int[TAB_SIZE];
}
```

☐
```
class EvilCode {
  abstract void dizzy() {
    System.out.println("boo");
  }
}
```

☐
```
package telecomnancy;
include java.util.List;
class EvilCode {
  void call() {}
  List<Integer> numbers;
}
```

☒
```
package eu.telecomancy.poo;
import java.util.*;
class EvilCode {
  void cast() {}
}
```

☒
```
package telecomnancy;
class EvilCode {
  Integer count = 9;
  void compute() {}
}
```

☐
```
package telecomnancy;
class EvilCode {
  void cry() {}
  java.util.List<int> numbers;
}
```

▷ **Question 2.** En considérant le code suivant, indiquer les affirmations correctes lors de l'exécution de la méthode `arrangeFlowers()`. (2 réponses)

```
class Flower {
  public void fragrance() { System.out.println("Flower"); }
}
class Rose extends Flower {
  public void fragrance() { System.out.println("Rose"); }
}
class Bouquet {
  public void arrangeFlowers() {
    Flower f1 = new Rose();
    f1.fragrance();
  }
}
```

☐ L'affichage produit est : `Flower`.
☒ L'affichage produit est : `Rose`.
☐ Le type dynamique de `f1` est `Flower`.
☒ Le type dynamique de `f1` est `Rose`.
☐ Le type statique de `f1` est `Rose`.

▷ **Question 3.** En considérant la signature de méthode `public String speak(int age, String message)`, indiquer les définitions qui permettent de *surchager* correctement cette méthode. (3 réponses)

☐ `public String speak(int volume, String message)`
☐ `private String speak(int age, String message)`
☐ `public String talk(int age, String message)`
☒ `public String speak(int age, String message, int volume)`
☐ `public boolean speak(int age, String message)`
☒ `public String speak(String message, int age)`
☒ `public boolean speak(String message, int age, int volume)`

▷ **Question 4.** En considérant le code suivant, indiquer quelles sont les déclarations correctes qui peuvent être insérées à la place du marqueur `// INSERT CODE HERE`. (3 réponses)

```
interface Hackable {
  void hack();
}
class Laptop {
  public void hack() { System.out.println("Laptop hacked"); }
}
class Smartphone implements Hackable {
  public void hack() { System.out.println("Smartphone hacked"); }
}
class Test {
  // INSERT CODE HERE
  {
    o.hack();
  }
}
```

☒ `void tryToHack(Hackable o)`
☒ `void tryToHack(Laptop o)`
☒ `void tryToHack(Smartphone o)`
☐ `void tryToHack()`

▷ **Question 5.** En considérant le code suivant, indiquer quelle instruction insérée à la place du marqueur `/* INSERT CODE HERE */` permet d'afficher la valeur de la variable `solarCellCount`. (1 réponse)

CORRECTION CORRECTION

```
class Robot {
  int power;
}
class SolarRobot extends Robot {
  int solarCellCount;
  public static void main(String args[]) {
    Robot myRobot = new SolarRobot();
    System.out.println(/* INSERT CODE HERE */);
  }
}
```

- ☐ `myRobot.solarCellCount`
- ☐ `SolarRobot.myRobot.solarCellCount`
- ☐ `(SolarRobot) myRobot.solarCellCount`
- ☒ `((SolarRobot) myRobot).solarCellCount`

▷ **Question 6.** Indiquer l'affichage produit par l'exécution du code suivant. (1 réponse)

```
class Plant {
  static void watering() {
    System.out.println("Watering a plant");
  }
}
class Cactus extends Plant {
  static void watering() {
    System.out.println("Watering a cactus");
  }
}
class TestWatering {
  public static void main(String[] args) {
    Plant plant = new Plant();
    Plant cactus = new Cactus();
    plant.watering();
    cactus.watering();
  }
}
```

- ☒ `Watering a plant`
`Watering a plant`
- ☐ `Watering a plant`
`Watering a cactus`
- ☐ `Watering a cactus`
`Watering a plant`
- ☐ Erreur à la compilation

▷ **Question 7.** Considérant les définitions de la classe `Season` et de l'interface `Sunny`, la tâche est de déclarer une classe `Summer` qui hérite de la classe `Season` et qui réalise l'interface `Sunny`. Indiquer la définition correcte. (1 réponse)

```
class Season {}
interface Sunny {}
```

- ☐ `class Summer implements Season extends Sunny {}`
- ☒ `class Summer extends Season implements Sunny {}`
- ☐ `class Summer implements Season, Sunny {}`
- ☐ `class Summer extends Season, Sunny {}`

▷ **Question 8.** En considérant les classes définies ci-dessous, indiquer quelles instructions de code peuvent être insérées individuellement à la place du marqueur `//INSERT CODE HERE` afin que l'exécution du code produise l'affichage suivant : (2 réponses)

```
Student says good lecture!
Gamer says good game!
```

```
class Student {
  void print() {
    System.out.println("Student says good lecture!");
  }
}
class Gamer extends Student {
  void print() {
    System.out.println("Gamer says good game!");
  }
}
class MyApp {
  Student a = new Student();
  // INSERT CODE HERE
  a.print();
  b.print();
}
```

- ☒ `Gamer b = new Gamer();`
- ☐ `Student b = new Student();`
- ☐ `Gamer b = new Student();`
- ☒ `Student b = new Gamer();`
- ☐ `Gamer b = ((Gamer) new Student());`
- ☐ `Student b = ((Gamer) new Student());`

▷ **Question 9.** Indiquer l'affichage produit par l'exécution du code suivant. (1 réponse)

CORRECTION

```
1  class DVD {
2    String title;
3    DVD(String t) {
4      title = t;
5    }
6  }
7  class TestEquals {
8    public static void main(String[] args) {
9      String name = "Game of Thrones";
10     DVD disc1 = new DVD(name);
11     DVD disc2 = new DVD(name);
12     System.out.print(disc1.equals(disc2) +":");
13     System.out.print(disc1 == disc2);
14   }
15 }
```

- ☐ `true:true`
- ☒ `false:false`
- ☐ `true:false`
- ☐ `false:true`
- ☐ Erreur à la compilation.
  Aucune méthode `equals` n'est définie dans la classe `DVD`.

▷ **Question 10.** En considérant le code suivant, indiquer quelle instruction insérée à la place du marqueur `/* INSERT CODE HERE */` permettrait à la classe `PokeDex` de déterminer si la variable `pikachu` référence un objet de la classe `ElectricPokemon` et d'afficher `"PikaPika"` dans ce cas. (1 réponse)

```
1  class Pokemon { float power; }
2  class ElectricPokemon extends Pokemon {
3    boolean electric;
4  }
5  class PokeDex {
6    public static void main(String args[]) {
7      Pokemon pikachu = new ElectricPokemon();
8      /* INSERT CODE HERE */
9        System.out.println("PikaPika");
10   }
11 }
```

- ☒ `if (pikachu instanceof ElectricPokemon)`
- ☐ `if (pikachu.instanceOf(ElectricPokemon))`
- ☐ `if (pikachu == ElectricPokemon)`
- ☐ `if (pikachu = ElectricPokemon)`

▷ **Question 11.** L'exécution de la méthode `main` de la classe `TestSensor` produit l'affichage `19.`. Indiquer quel serait l'affichage produit si la variable `temperature` était définie en tant que variable de classe. (1 réponse)

```
1  class Sensor {
2    double temperature;
3  }
4  class TestSensor {
5    public static void main(String args[]) {
6      Sensor s1 = new Sensor();
7      Sensor s2 = new Sensor();
8      s1.temperature = 18.;
9      s2.temperature = 20.;
10     System.out.println((s1.temperature + s2.temperature)/2);
11   }
12 }
```

- ☐ L'exécution de `TestSensor` affiche `18.`
- ☐ L'exécution de `TestSensor` affiche `19.`
- ☒ L'exécution de `TestSensor` affiche `20.`
- ☐ La classe `TestSensor` ne compile plus.

▷ **Question 12.** En considérant les définitions de classes suivantes, quels morceaux de code insérés individuellement à la place du marqueur `// INSERT CODE HERE` définiront une utilisation valide de la méthode `fly()` dans la méthode `takeATrip()`. (3 réponses)

```
1  class NoMoreFuelException extends Exception {}
2  class Plane {
3    void fly(long distance) throws NoMoreFuelException {
4      //.. some code
5    }
6    void takeATrip() {
7      // INSERT CODE HERE
8    }
9  }
```

☒
```
try {
  new Plane().fly(1200);
} catch (NoMoreFuelException e) {}
```

☐
```
try {
  (new Plane()).fly(1500);
} catch (RuntimeException e) {}
```

☒
```
try {
  fly(900);
} catch (Exception e) {}
```

☒
```
try {
  (new Plane()).fly(1100);
} catch (NoMoreFuelException e) {
} finally { }
```

☐
```
try {
  (new Plane()).fly(800);
}
```

☐
```
try {
  (new Plane()).fly(700);
} finally {}
```

CORRECTION

▷ **Question 13.** Indiquer les réponses correctes.

Correcte ?	
☐	Une classe est une instance d'un objet.
☒	Dans une méthode, il est possible d'accéder à l'instance courante à l'aide du mot-clé `this`.
☒	Le mot-clé `private` permet de garantir qu'une variable ne pourra être accédée que par toutes les instances de la classe à laquelle elle appartient.
☒	Il est possible de déclarer des variables dans une interface si elles sont déclarées `final`.
☐	Il est impossible d'appeler un constructeur à partir d'un autre constructeur de la même classe.
☐	Il faut toujours définir un constructeur sans paramètre dans une classe.
☐	Une classe mère ne peut pas définir de membres protected.
☐	Une classe fille a accès à tous les membres de ses super-classes.
☒	En Java, toutes les classes héritent de la classe `Object`.
☐	Les variables statiques sont accessibles uniquement à partir de méthodes statiques.
☐	Une classe abstraite ne peut hériter d'une autre classe abstraite.
☐	Une classe abstraite doit définir au moins une méthode abstraite.
☒	Une méthode abstraite peut être appelée dans la classe qui la définit.
☒	Les exceptions sont des objets.
☒	Les méthodes de la classe `String` ne modifient jamais le contenu du receveur.

▷ **Question 14.** Expliquer la différence entre le concept de liaison dynamique et celui de liaison statique. Illustrer votre propos par deux exemples de code (un pour chaque type de liaison).

CORRECTION

▷ **Question 15.** Expliquer la différence entre une exception vérifiée et une exception non-vérifiée. Indiquer dans quel cas il est convient d'utiliser l'une ou l'autre.

CORRECTION

▷ **Question 16.** Expliquer la notion de *couplage* et pourquoi il est important de chercher à le minimiser en conception objet.

CORRECTION

▷ **Question 17.** Donner deux mesures qu'il est recommandé de mettre en place pour préserver l'encapsulation au sein d'une classe. Illustrer vos propos en donnant un exemple de code java.

▷ **Question 18.** Expliquer quel peut être l'intérêt de définir une classe abstraite.

CORRECTION

CORRECTION