Classes abstraites et interfaces

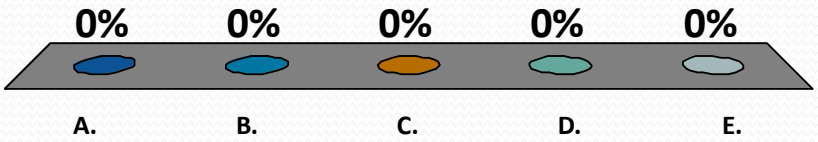
Christelle CAILLOUET
(christelle.caillouet@unice.fr)

Rappels

Quelle est la relation qui lie les classes `Point` et `Segment` ?

✓ A. Composition
B. Encapsulation
C. Héritage
D. Instanciation
E. Polymorphisme

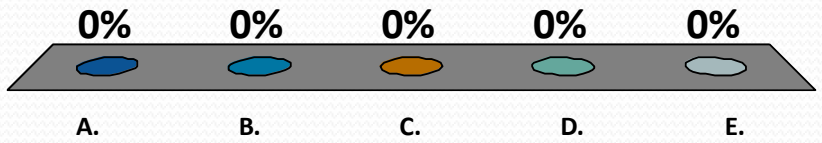
30



Quelle est la relation qui lie les classes `Point` et `ColoredPoint` ?

A. Composition
B. Encapsulation
✓ C. Héritage
D. Instanciation
E. Polymorphisme

30



Pourquoi…

- Un objet `ColoredPoint` n'est-il pas plutôt composé d'un point et d'une couleur ?

```
public class ColoredPoint {
    private Point p;
    private Color couleur;
    …
```

Rappels

- **Composition** : relation d'appartenance « *has* »

- **Héritage** : relation d'existence « *is* »

➔Un `ColoredPoint` est avant tout un `Point`

➔Un point coloré doit avoir les mêmes fonctionnalités qu'un point (puisqu'il en est un)

Rappels

- **Composition** : relation d'appartenance « *has* »

- **Héritage** : relation d'existence « *is* »

➔A l'inverse, un `Segment` **n'est pas un** `Point`

➔Un `Segment` **possède 2** `Point`(s)

Classes abstraites

Classe abstraite

- Une classe abstraite **n'est pas instanciable**

- Elle permet de :
  - Définir des constantes et/ou des services utilisables par d'autres classes
  - Définir des méthodes abstraites (sans code)

- Elle ne peut servir que de classe de base pour une dérivation

Intérêt d'une classe abstraite

- Définir des membres (attributs et méthodes) communs à une hiérarchie de classes
  - Factorisation d'attributs et de méthodes
- Faciliter la conception et le code
  - La classe abstraite regroupe toutes les fonctionnalités pour ses classes dérivées (*descendantes*)
- Permet d'exploiter le polymorphisme

Définition d'une classe abstraite

- Utilisation du mot-clé **abstract**

```
public abstract class A {
    public void f() { ... }
    public abstract void g(int n);
}
```

Méthode *f* implémentée

Méthode *g* **abstraite** (déclarée dans la classe mais pas implémentée)

Type classe abstraite

- La déclaration `A a` est autorisée
- L'instanciation `a = new A();` est interdite *(même si un constructeur est présent)*

Classe abstraite et héritage

```
public abstract class A {
    public void f() { ... }
    public abstract void g(int n);
}

public class B extends A {
    public B() {...}
    public void g(int n) {...}
}
```

- `A a = new B();` ➔ **Autorisé !**

Classe abstraite et héritage

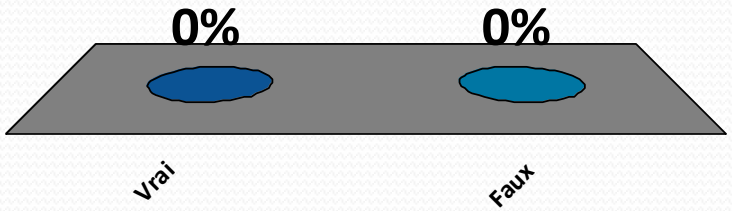
- Lorsqu'une classe hérite d'une classe abstraite, elle doit :
  - Soit implémenter les méthodes abstraites de sa super classe en les dotant d'un corps
  - Soit être elle-même abstraite si au moins une des méthodes abstraites de sa super classe reste abstraite

Une classe dérivée d'une classe non abstraite peut-elle être abstraite ?

✓ A. Vrai

B. Faux

30



Une classe dérivée d'une classe non abstraite peut-elle être abstraite ?

- Toute classe dérive de la classe `Object` qui n'est pas abstraite…

Interfaces

Interface

- C'est une classe « *entièrement* » abstraite
- Elle n'implémente aucune méthode ni aucun champ (hormis des constantes) ~~> méthodes *default* (Java 8)
- On y définit les en-têtes d'un certain nombre de méthodes ainsi que des constantes

Intérêt des interfaces

- Comme les classes abstraites :
  - Les interfaces peuvent se dériver
  - On peut utiliser des variables de type interface

- De plus :
  - Une classe peut implémenter plusieurs interfaces (*une classe ne pouvait dériver que d'une classe abstraite)*
  - La notion d'interface se superpose à celle de dérivation (héritage)

➔ Sert à donner un type commun à des classes différentes pour en faire un même usage

Définition d'une interface

- Le mot-clé **interface** **se substitue à class**

```
public interface Affichable {
    String toString();
}

interface Deplacable {
    void translation(double dx, double dy) throws Exception;
}

interface Pivotable {
    static final double PI=3.14159 ;
    void rotation(double angle);
}
```

Propriétés

- Les droits d'accès d'une interface sont régis par les mêmes règles que ceux d'une classe (public ou par défaut droit dans le package)
- Toutes les méthodes d'une interface sont implicitement ***publiques*** et ***abstraites***
- Tous les champs d'une interface sont implicitement des constantes (***public final static*** ajouté implicitement)
- L'interface permet de définir un *contrat*
  - Que doivent remplir les classes qui en « héritent »
  - À travers une ou plusieurs méthodes

Implémentation d'une interface

- Lorsqu'une classe implémente une interface, utilisation du mot-clé **implements**

```
public class A implements Affichable
```

- La classe doit définir les méthodes prévues dans l'interface
  - Sinon erreur de compilation

```
public class A implements Affichable {
    public void affiche() {
        System.out.println("Classe A");
    }
}
```

Implémentation d'une interface

- Une même classe peut implémenter plusieurs interfaces

```
public class A implements Affichable, Deplacable
```

- L'implémentation d'une interface est complètement indépendante de l'héritage

Héritage et interface

- On peut définir une interface comme une généralisation d'une autre

```
interface I1 {
    void f(int n);
    static final int MAXI = 100;
}

interface I2 extends I1 {
    void g();
    static final int MINI = 20;
}
```

➔

```
interface I2 {
    void f(int n);
    void g();
    static final int MAXI = 100;
    static final int MINI = 20;
}
```

- La dérivation d'interface revient à concaténer les déclarations

Héritage et interface

- Une classe peut **hériter** d'une classe de base et **implémenter** plusieurs interfaces

- B doit implémenter les méthodes des interfaces

```
public class B extends A implements I1, I2
```

Que se passe-t-il si 2 méthodes de même nom et même signature de différentes interfaces doivent être implémentées dans la même classe ?

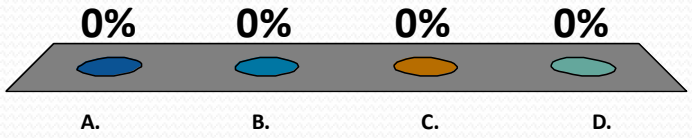
A. Erreur de compilation

B. Erreur à l'exécution (ambiguité)

C. ✓ Une seule implémentation valable pour les 2 méthodes

D. Surcharger les 2 méthodes

60



Conflit de noms

```
interface I1 {
    void f(int n);
    void g();
}

interface I2 extends I1 {
    void f(float x);
    void g();
}

class A implements I1, I2 {
    // A doit définir 2 méthodes f : void f(int) et void f(float)
    // mais une seule méthode g
}
```

Interface et polymorphisme

- On peut définir des variables de type interface

```
public interface Affichable {…}
public class A implements Affichable {…}

Affichable a;
```

- On peut affecter à un objet de type interface une référence vers un objet d'une classe implémentant l'interface

```
a = new A();
```

Interface et polymorphisme

- A travers l'interface, on peut manipuler des objets de classe quelconque, non nécessairement liées par héritage

- Le tout étant que la classe implémente l'interface.

Nouveauté Java 8

- Java 8 élargit le concept d'interface :
  - Possibilité d'y définir des méthodes *par défaut* (mot-clé **default**)
  - Ces méthodes disposent d'une implémentation

- Une méthode **default** :
  - Peut être utilisée directement par une classe implémentant l'interface
  - Peut être redéfinie dans une classe ou interface dérivée

Design : interface ou héritage

- On **hérite** d'une classe :
  - Pour créer un nouveau type qui est « une sorte particulière » de la classe de base

- On définit une **interface** et on l'implémente :
  - Pour une fonctionnalité transverse (`Comparable`, `Mesurable`, `Déplaçable`, `Closeable`, …)
  - Pour regrouper un ensemble de fonctionnalités qui pourront être implémentées par des instances qui en implantent déjà d'autres (ou qui héritent d'une autre classe)

```
public class RandomAccessFile extends Object
implements DataOutput, DataInput, Closeable {…}
```

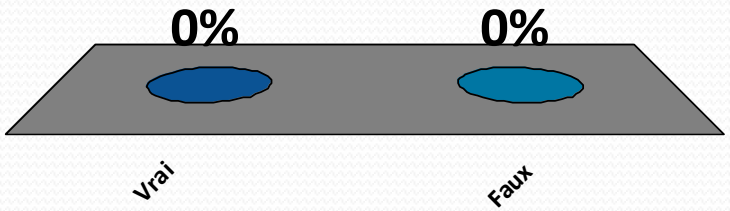
Questions

Une classe abstraite peut contenir des attributs (variables) ?

✓ A. Vrai

B. Faux

20

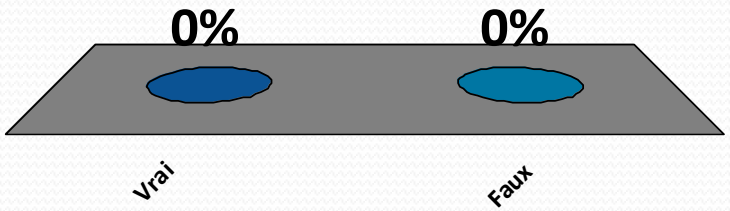


Une interface peut contenir des attributs (variables) ?

A. Vrai

✓B. Faux

20

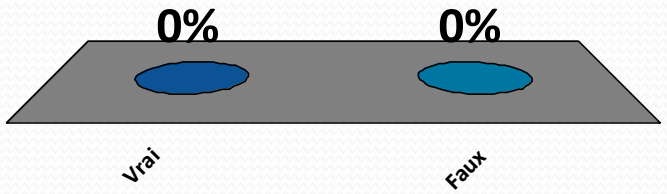


Une classe qui dérive d'une classe abstraite peut-elle laisser une méthode abstraite non implémentée ?

✓ A. Vrai

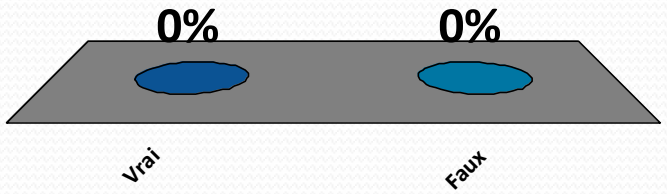
B. Faux

20



Une classe qui implémente une interface peut-elle laisser une méthode de l'interface non implémentée ?

A. Vrai

✓ B. Faux

20