STROBBE Nathan
Mark: 15/20 (total score: 18/24)

+26/1/47+

QCM **TEST**

Introduction à la programmation orientée objet
10/01/2018

Nom et prénom : STROBBE Nathan

Groupe : 2

Cochez les cases en mettant une ×.
Le symbole ⊕ indique que la question peut avoir zéro, une ou plusieurs bonnes réponses. Pour ces questions, cocher une bonne réponse apporte des points positifs ; cocher une mauvaise réponse peut apporter des points négatifs.
Dans tout le code, les `package` et les `import` sont censés être correctement déclarés. Toute classe est supposée être dans le bon package, dans le bon fichier, avec les bons import.

Question 1 ⊕ Soit les classes suivantes:

```
interface Voiture {
  void drive();
}
```

```
class DeLoreanDMC12 {
    private void drive() {
      System.out.println("Je peux la conduire");
    }
}
```

```
class MartysDMC extends DeLoreanDMC12 implements Voiture {
    public void drive() {
      System.out.println("Je peux la conduire et aussi la faire voler");
    }
}
```

Si j'écris `new MartysDMC().drive()` :

- ☐ Mon programme affichera `Je peux la conduire`.
- ☐ Mon programme ne se compilera pas car le compilateur se plaint que la classe DeLoreanDMC12 n'implémente pas l'interface Voiture.
- ☐ Mon programme ne se compilera pas car les droits d'accès de la méthode drive() ne peuvent être redéfinis dans la classe MartysDMC.
- ☒ Mon programme affichera `Je peux la conduire et aussi la faire voler`.
- ☐ Mon programme s'arrêtera à l'exécution faute de droits d'accès suffisants.

2/2

Question 2 Lors d'une expédition archéologique, les scientifiques ont ramené le code ci-dessous d'un site Néanderthal :

```
package instanceov;

class Apple {}
class Banana {}
class Kiwi {}
class Apteryx {}
class Hamster {}

public class Miam {
  private Object[] mets = {new Kiwi(), new Hamster(), new Apple(),
                           new Banana(), new Apteryx()};
  public void manger(Object obj) {
    if (obj instanceof Apple) {
      System.out.println("J'epluche et je mange une pomme");
    } else if (obj instanceof Banana) {
      System.out.println("J'epluche et je mange une banane");
    } else if (obj instanceof Kiwi) {
      System.out.println("J'epluche et je mange un kiwi");
    } else if (obj instanceof Apteryx) {
      System.out.println("J'epluche [sic], je fais cuire et je mange un apteryx");
    } else if (obj instanceof Hamster) {
      System.out.println("J'epluche [sic], je fais cuire et je mange un hamster");
    }
  }
  public static void main(String[] args) {
    Miam miam = new Miam();
    for (Object met : miam.mets) {
      miam.manger(met);
    }
  }
}
```

À l'exécution le code affiche:

```
J'epluche et je mange un kiwi
J'epluche [sic], je fais cuire et je mange un hamster
J'epluche et je mange une pomme
J'epluche et je mange une banane
J'epluche [sic], je fais cuire et je mange un apteryx
```

ce qui indiquerait qu'il fonctionne correctement.
A part l'orthographe approximative, en quoi peut-on facilement reconnaître qu'il s'agit d'un code primitif ? (Penser à l'introduction d'un nouveau mets au menu, de l'orignal par exemple)

☐0 ☐1 ☒3 0.5/0.5

On utilise ici beaucoup de "instance of" ce qui n'est pas très orienté objet. Il y a un problème de conception car on regarde chaque type dynamique de chaque objet.

Question 3 Les difficultés à maintenir le code ci-dessus ont certainement contribué à l'extinction de l'homo sapiens neandertalensis. En tant que sapiens sapiens, refactorer (c'est à dire modifier tout en gardant la fonctionnalité) ce code pour être plus facilement recyclable par votre descendance.

☐0 ☒1 ☐3

0.33/0.5

Il suffirait de mettre un toString() pour les classes Apple, Banane, Kiwi, Apteryx, Hamster. Et de faire un appel : System.out.println(food); Avec food paramètre de manger(Object food)

On pourrait même faire une interface Comestible pour éviter d'utiliser Object qui est trop générique.

Question 4 Pour le code primitif original, quel serait l'effet de `new Miam().manger(new Caillou());` où la classe Caillou est déjà définie ?

☐0 ☐1 ☒3

0.5/0.5

Aucun effet, ici on peut manger un caillou et rien ne se passe.

Question 5 Toujours pour le code primitif original, comment indiquer qu'un objet de la classe *Caillou* n'est pas vraiment comestible ? Un traitement par Exception vous semble-t-il indiqué ?

☐0 ☒1 ☐3

0.33/0.5

On pourrait avoir un attribut estComestible dans la classe Caillou. (avec un getter)
On peut gérer sa comestibilité par Exception lors de l'appel de manger.

```
public void manger(Object food) throws NonComestibleException {
    if (!food.isComestible())
        throw new NonComestibleException(food);
    System.out.println(food);
}
```

Question 6 Que ce soit indiqué ou pas, rajouter ce qu'il faut pour avoir un traitement **exceptionnel** d'une tentative de manger un *Caillou*.

☐0 ☐1 ☐3

0/0.5

```
try {
    for(Object met : miam.mets) {
        miam.manger(met);
    }
} catch(NonComestibleException e){
    System.out.println(e.toString()+" n'est pas comestible!");
}
```

Pour toutes les questions suivantes, cochez toutes les cases où le résultat du remplacement de `____` par le niveau d'accès correspondant est légal (cela compile).

Question 7 ⊕

```
____ class Toto {}
```

☐ private
☒ *package*
☒ public
☐ protected

0.25/0.25

Question 8 ⊕

```
class Toto {
  ____ String s;
}
```

0.25/0.25

- [x] public
- [x] private
- [x] *package*
- [x] protected

Question 9 ⊕

```
class Toto {
  ____ void m() {}
}
```

0.25/0.25

- [x] private
- [x] public
- [x] *package*
- [x] protected

Question 10 ⊕

```
class Toto {
  ____ static String s;
}
```

0.25/0.25

- [x] *package*
- [x] private
- [x] protected
- [x] public

Question 11 ⊕

```
void m() {
  ____ int i = -17;
}
```

0.25/0.25

- [ ] private
- [ ] public
- [ ] protected
- [x] *package*

Question 12 ⊕

```
abstract class Toto {
  ____ void m();
}
```

0.12/0.25

- [x] protected
- [ ] private
- [x] public
- [x] *package*

Question 13 ⊕

```
____ interface Toto {}
```

0.25/0.25

- [x] *package*
- [ ] protected
- [x] public
- [ ] private

Question 14 ⊕

```
interface Toto {
  ____ void m();
}
```

0.25/0.25

- [ ] private
- [ ] protected
- [x] public
- [x] *package*

Question 15 ⊕

```
____ static void main(String[] a) {}
```

- [ ] protected
- [ ] *package*
- [x] public
- [ ] private

0.25/0.25

Question 16 ⊕

```
class Toto {
  ____ class Titi {}
}
```

- [x] private
- [x] public
- [x] protected
- [x] *package*

0/0.25

Question 17 ⊕

```
class Toto {
  class Titi {
    ____ String s;
  }
}
```

- [x] public
- [x] private
- [x] protected
- [x] *package*

0.25/0.25

Question 18 ⊕

```
interface Toto {
  void m();
}

class Titi {
  Toto t = new Toto() {
    ____ void m() {}
  };
}
```

- [x] *package*
- [ ] private
- [x] public
- [ ] protected

0.12/0.25

Question 19 ⊕
On modélise le comportement d'archers célèbres dans les classes suivantes :

```
class Archer {
  void tirer() {
    System.out.println("Une fleche dans le mille !");
  }
}
```

```
class NasuYoichi extends Archer {
  void tirer() {
    System.out.println("Atari desu !");
  }
}
```

```
class GuillaumeTell extends Archer {
  void tirer() {
    System.out.println("Pom pom pom pom !");
  }
}
```

Selon vous, le code:

```
class Main {
  public static void main(String[] args) {
    Archer a = new GuillaumeTell();
    a.tirer();
    NasuYoichi n = (NasuYoichi) a;
    n.tirer();
  }
}
```

- [x] va générer une exception à l'exécution
- [ ] va afficher `Atari desu !`
- [ ] ne va pas compiler
- [ ] va afficher `Une fleche dans le mille !`
- [x] va afficher `Pom pom pom pom !`

1.2/2

Question 20 La classe suivante:

```
class Alpine {
    Alpine() {}

    @Override
    private void displayModel() {
        System.out.println("Alpine A110");
    }
}
```

ne peut être compilée alors qu'elle se trouve dans le répertoire courant. Quelle en est la raison la plus évidente ?

- [ ] Le compilateur ne connaît pas l'emplacement de Alpine.java.
- [ ] Le constructeur est vide.
- [x] La classe n'hérite d'aucune classe.
- [ ] La classe Alpine n'a pas importé les packages qui lui manquent pour afficher sur la console.
- [x] Les droits d'accès à la méthode *displayModel* sont insuffisants.

0/2

Vehicule v;
Car c = new Car();
v = c;
c = v;
c = (Car) v; si v est un Car

Question 21 ⊕

Un programmeur débutant dispose des deux classes suivantes:

```
class Original {
  int x;
  Original(int x) { this.x=x; }
  void print() { System.out.println("x="+x); }
}
```

```
class Extender extends Original {
  int y;
  Extender(int x, int y) { super(x); this.y=y; }
  void print() { System.out.println("x="+x+", y="+y); }
}
```

Il décide de les employer dans le programme suivant :

```
import java.util.ArrayList;
import java.util.List;
public class Main {
  static void display(List l) {
    for(int i=0;i<l.size();i++)
      ((Original)l.get(i)).print();
  }
  public static void main(String[] args) {
    Original o1 = new Original(3);
    Original o2 = new Original(7);
    List<Original> lo = new ArrayList<Original>();
    lo.add(o1); lo.add(o2);
   display(lo);
  }
}
```

qui affiche alors:

```
x=3
x=7
```

Notre programmeur souhaite maintenant écrire un nouveau programme affichant les coordonnées d'objets de classe Extender stockés dans des listes comme suit:

```
public static void main(String[] args) {
  Extender e1 = new Extender(2,4);
  Extender e2 = new Extender(6,7);
  Extender e3 = new Extender(3,9);
  XXX le = new YYY();
  le.add(e1); le.add(e2); le.add(e3);
  display(le);
}
```

Quelles affirmations sont **vraies** parmi les phrases suivantes ?

- ☐ `XXX=List<Original>` et `YYY=ArrayList<Original>` provoque une erreur de typage à l'exécution
- ☒ `XXX=List<Object>` et `YYY=ArrayList<Object>` affiche correctement les coordonnées
- ☒ `XXX=List<Original>` et `YYY=ArrayList<Original>` affiche correctement les coordonnées
- ☒ `XXX=List<Extender>` et `YYY=ArrayList<Extender>` affiche correctement les coordonnées
- ☐ `XXX=List<Extender>` et `YYY=ArrayList<Extender>` provoque une erreur de typage à l'exécution
- ☐ `XXX=List<Object>` et `YYY=ArrayList<Object>` provoque une erreur de typage à l'exécution

1/1

Question 22 Comment pourrait-on détecter une erreur à la compilation pour empêcher un plantage à l'exécution en modifiant le code de la méthode ***display*** et quelles en seraient les conséquences?

☐0 ☐1 ☒3 1/1

Il faudrait définir le type exact de la List en paramètre de display :
static void (List<Original> l) {}

La conséquence est qu'il faudrait appeler cette méthode avec une list d'Original ou Extender

Question 23 ⊕

Un programmeur débutant a écrit le code suivant :

```
interface Forme {
  public void afficherCoordonnees();
  public void afficherType();
}
```

```
class Figure {
  public static int count = 0;
  int x,y;
  Figure() {count++;}
  Figure(int x, int y) {count++; this.x=x; this.y=y;}
  void ajouterCoordonnees(int x, int y) {
        this.x=x;
    this.y=y;
  }
  public void afficherCoordonnees() {
    System.out.print("x="+x+";y="+y);
  }
  public void afficherType() {
         System.out.print("figure");
  }
}
```

```
class Carre extends Figure {
  Carre() {
    super();
  }
  Carre(int x, int y) {
        super(x,y);
  }
  public void afficherType() {
    System.out.print("carre");
  }
}
```

```
class Cercle extends Figure implements Forme {
  Cercle() {
    super();
  }
  Cercle(int x, int y) {
        super(x,y);
  }
  public void afficherType() {
    System.out.print("cercle");
  }
}
```

```
class Main {
  static void displayForme(Forme f) {
    System.out.print("coordonnees ");
    f.afficherType();
    System.out.print(" : ");
    f.afficherCoordonnees();
    System.out.println(".");
  }
  public static void main(String[] args) {
    Cercle ce = new Cercle(4,5);
    Carre ca = new Carre(3,2);
    System.out.println("nombre de cercles="+ce.count);
    Figure fi1 = ce;
    Forme fo1 = (Forme) fi1;
    displayForme(fo1);
    Figure fi2=ca;
    Forme fo2 = (Forme) fi2;
    displayForme(fo2);
  }
}
```

Quelles affirmations sont **vraies** parmi les phrases suivantes ?

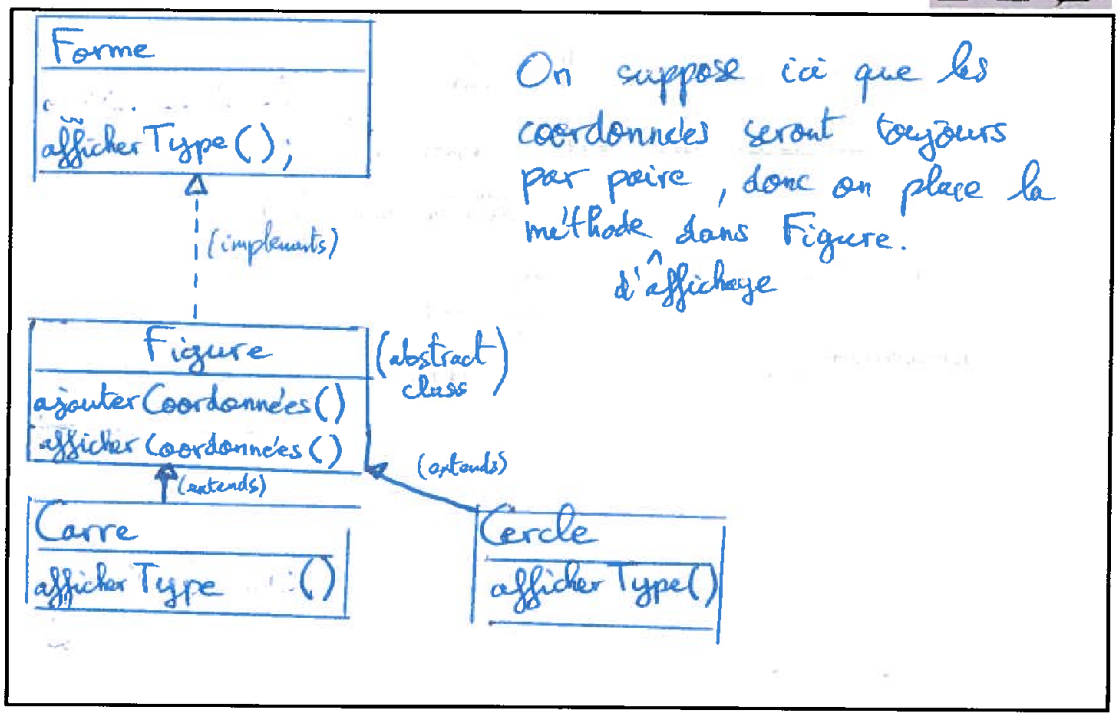
- ☐ Un objet de type *Carre* est de type interface *Forme* (par polymorphisme)
- ☒ Un objet de type *Figure* implémente les méthodes définies dans l'interface *Forme*
- ☒ Le code ne compile pas car l'attribut *count* de la classe Figure ne peut être hérité à cause de la déclaration `static`
- ☐ le programme va afficher `coordonnees carre : x=3 ;y=2.`
- ☐ Le code ne compile pas pour cause d'erreur de typage
- ☒ Un objet de type *Carre* est de type *Figure* (par polymorphisme)
- ☐ Le code ne compile pas car la méthode *displayForme* ne peut être déclarée `static` dans la classe *Main*
- ☒ le programme va afficher `coordonnees cercle : x=4;y=5.`

0.5/2

Question 24 Comment améliorer l'architecture utilisée en réorganisant les classes et interfaces (Forme, Figure, Cercle, Carre ...) ?

☐0 ☐1 ☒3

1/1



Question 25 ⊕

Soit le programme suivant :

```
import java.util.ArrayList;
import java.util.List;
class Tableau<T> {
  List<T> monTableau;
  Tableau(List<T> tableauInitial) {
     monTableau=tableauInitial;
  }
  void afficher(int index) {
     try{
       for (int i=0; i<=index; i++) {
          System.out.print(i/monTableau.get(i)+";");
       }
     }
     catch(RuntimeException e) {
       System.out.println("An error occurred in
         Tableau.afficher: "+e);
     }
     catch(Exception e) {
       System.out.println("An error occurred in
         Tableau.afficher: "
       +new ApplicationException());
     }
     }
  }
}
```

```
class ApplicationException extends Exception {}
```

```
import java.util.ArrayList ;
import java.util.List ;
class Main {
  public static void main(String[] args) {
    List<Integer> l=new ArrayList<Integer>();
    l.add(0); l.add(1); l.add(2); l.add(3);
    Tableau t=new Tableau(l);
    try {
      t.afficher(4);
    }
    catch(Exception e) {
      System.out.println("An error occurred in
        Main.main: "+e);
    }
    System.out.println("Termine !");
  }
}
```

Quels sont les résultats de la liste suivante qui apparaissent lorsque ce code est exécuté ?

- ☐ `An error occurred in Main.main: ApplicationException`
- ☐ `An error occurred in Main.main: java.lang.IndexOutOfBoundsException: Index: 4, Size: 4`
- ☒ `Termine !`
- ☐ `An error occurred in Tableau.afficher: java.lang.IndexOutOfBoundsException: Index: 4, Size: 4`
- ☒ `An error occurred in Tableau.afficher: java.lang.ArithmeticException: / by zero`
- ☐ le code ne compile pas car le compilateur indique: `Tableau.java:15: error: unreported exception ApplicationException; must be caught or declared to be thrown`
- ☐ `An error occurred in Main.main: java.lang.ArithmeticException: / by zero`
- ☐ `An error occurred in Tableau.afficher: ApplicationException`

2/2

Question 26 Quelle serait selon vous la meilleure manière de rattraper une erreur d'introduction des données par le programmeur dans ce même exemple ? ☐0 ☐1 ☒3

1/1

Le développeur de la méthode devrait vérifier si en paramètre il reçoit un entier nul car il fait une division et dans ce cas il throw new ArithmeticException il ne devrait pas utiliser de try/catch.
C'est le développeur du main qui doit gérer un try/catch

Question 27 ⊕

Soit les classes suivantes :

```
public class A {
  public void m(A a) {
    System.out.println("c'est un AA");
  }
  public void m(B b) {
    System.out.println("c'est un AB");
  }
}
```

```
public class B extends A {
  public void m(A a) {
    System.out.println("c'est un BA");
  }
  public void m(B b) {
    System.out.println("c'est un BB");
  }
}
```

```
class Main {
  public static void Main(String[] args) {
    A o1 = new A();
    A o2 = new B();
    o2.m(o1);
  }
}
```

```
\end{minipage}
\end{lrbox}

\newsavebox\qbMain
\begin{lrbox}{\qbMain}
\begin{minipage}{0.49\linewidth}
  {\scriptsize
  \begin{lstlisting}
public class Main {
  public static void main(String[] args) {
    A o1 = new A();
    A o2 = new B();
    o2.m(o1);
  }
}
```

Selon vous, que va-t-il se passer ?

1.5/1.5

- ☐ le programme va générer une exception à l'exécution
- ☐ le programme ne va pas compiler
- ☐ le programme va afficher `C'est un AA`
- ☒ le programme va afficher `C'est un BA`
- ☐ le programme va afficher `C'est un BB`
- ☐ le programme va afficher `C'est un AB`

Question 28 ⊕ Que se passera-t-il si toutes les méthodes m du code précédent sont déclarées static ?

- ☐ le programme va afficher `C'est un BA`
- ☒ le programme ne va pas compiler
- ☐ le programme va afficher `C'est un BB`
- ☐ le programme va afficher `C'est un AB`
- ☒ le programme va afficher `C'est un AA`
- ☐ le programme va générer une exception à l'exécution

0.5/1.5

Question 29 ⊕

On modélise les tournevis d'un atelier comme suit:

```
class Tournevis {
  static boolean occupe = false;
  void visser() {
    if (!occupe) {
      occupe=true;
      System.out.println("on visse ...");
    }
    else System.out.println("deja utilise !");
  }
  void reposer() {
    occupe=false;
    System.out.println("Tournevis repose et disponible");
  }
}
```

Quel affichage le code suivant produira-t-il ?

```
class Main {
    static public void main(String[] args) {
        Tournevis t1 = new Tournevis();
        Tournevis t2 = new Tournevis();
        t1.visser();
        t1.visser();
        t2.visser();
        t1.reposer();
        t1.visser();
    }
}
```

- ☒
```
on visse ...
deja utilise !
deja utilise !
Tournevis repose et disponible
on visse ...
```
- ☐
```
on visse ...
deja utilise !
on visse ...
Tournevis repose et disponible
on visse ...
```
- ☐
```
on visse ...
on visse ...
on visse ...
Tournevis repose et disponible
on visse ...
```
- ☐
```
deja utilise !
deja utilise !
deja utilise !
Tournevis repose et disponible
deja utilise !
```
- ☐
```
deja utilise !
deja utilise !
deja utilise !
Tournevis repose et disponible
on visse ...
```

2/2