Question 1 (**)

On considère l'extrait de code suivant :

```
public static float sumFloat(List<? super Number> list) {
    float x = 0;
    for (Number y : list)
        x += y.floatValue();
    return x;
}
```

Non, cet extrait de code ne compile pas. La méthode `sumFloat` accepte comme argument une `List<T>`, où `T` est un super-type de `Number`. Les éléments de la liste ne peuvent pas donc implicitement être downcastés en `Number`

Question 2 (*)

Lorsqu'une classe B étend une classe A, est-ce que : (entourer les réponses correctes)

Réponses possibles :

- × toutes les instances de A sont aussi des instances de B
- → toutes les instances de B sont aussi des instances de A
- × les deux réponses précédentes sont vraies
- × les trois réponses précédentes sont fausses.

Question 3 (**)

L'interface fonctionnelle `ToIntBiFunction<T,U>` (du package `java.util.function`) contient pour unique méthode `int applyAsInt(T value1, U value2)`. On souhaite utiliser cette interface pour calculer le produit des tailles de deux chaînes de caractères. Pour cela, on considère le code ci-dessous :

```
XXXX lengthProduct = YYYY;
String s1 = "hello"; String s2 = "world";
System.out.println(lengthProduct.applyAsInt(s1,s2)); // affiche 25
```

Que faut-il indiquer à la place de XXXX et YYYY pour que le fragment de code ci-dessus affiche le produit des tailles de `s1` et `s2` (25 en l'occurrence) ? Indice : YYYY doit contenir une expression lambda.

`ToIntBiFunction<String, String> lengthProduct = (s1, s2) -> s1.length()*s2.length();`

Question 4 (*)

```
public static void main(String args[]) {
    try {
        int a, b; b = 0; a = 5 / b;
        System.out.print("B ");
    } catch(ArithmeticException e) {
        System.out.print("A ");
    }
}
```

Qu'est-il affiché à l'exécution du programme ?

- → A
- × B
- × Exception in thread "main" java.lang.ArithmeticException : division by zero
- × B A

L'exception est correctement gerée, d'où l'affichage de 'A'.

Question 5 (***)

Écrire une expression lambda à la place du commentaire pour qu'à l'execution de la méthode `main`, le caractère `c` vaille `'3'`.

```
public interface Ab<A,B>{
  B ab(A a);
}
public class Barbara<A,B,C>{
  Ab<Ab<B,C>, Ab<Ab<A,B>, Ab<A,C>>> barbara =
  // Votre expression lambda ici g -> f -> x -> g.ab(f.ab(x))
}
public class Test {
  public static void main(String[] args) {
    Ab<Integer,String> f = i -> Integer.toString(i);
    Ab<String,Character> g = s -> s.charAt(0);
    Barbara<Integer,String,Character> barb =
      new Barbara<Integer,String,Character>();
    int test = 3345;
    char c = barb.barbara.ab(g).ab(f).ab(test);
    // c doit valoir '3'
  }
}
```

Question 6 (*)

```
public class B{}
public class A extends B{}
```

Réponses possibles :

- × toutes les méthodes de la classe `A` sont accessibles par la classe `B`,
- × `A` et `B` doivent appartenir au même package,
- × toutes les méthodes de la classe `A` sont héritées par la classe `B`,
- → la classe A peut déclarer une méthode avec la même signature qu'une méthode de la classe B.

Question 12 (*)

```
class B {
  int c = 0;
  public int getC () {
    return c;
  }
}
class A extends B {
  int c = 1;
}
public class Test {
  public static void main (String args[]) {
    A a = new A(); B b = a;
    System.out.print(b.getC()+" ");
    System.out.println(a.getC());
  }
}
```

Réponses possibles :

- × 1 1
- × 1 0
- × 0 1
- → 0 0

Question 7 (*)

```
class Portable {
   public void sonne(Sonnerie s) { s.sonne(); }
}
class Sonnerie {
   public void sonne() { System.out.println("Z Z"); }
}
class SimpleSonnerie extends Sonnerie{
   public void sonne() { System.out.println("Z"); }
}
public class Bruit {
  public static void main(String[] args) {
      Portable tel = new Portable();
      Sonnerie s1 = new Sonnerie();
      SimpleSonnerie s2 = new SimpleSonnerie();
      Sonnerie s3 = new SimpleSonnerie();
      tel.sonne(s1);
      tel.sonne((Sonnerie)s2);
      tel.sonne(s2);
      tel.sonne(s3);
  }
}
```

Réponses possibles :

- × Z Z [retour à la ligne] Z Z [retour à la ligne] Z [retour à la ligne] Z Z
- × Z Z [retour à la ligne] Z Z [retour à la ligne] Z [retour à la ligne] Z
- → Z Z [retour à la ligne] Z [retour à la ligne] Z [retour à la ligne] Z
- × Z [retour à la ligne] Z Z [retour à la ligne] Z Z [retour à la ligne] Z