Question 1 de 41

Quelles sont les modificateurs de visibilité en **Java** :

- **A.** public, private, sealed, transcient
- **B.** public, private, protected
- **C.** public, private, protected, package (par défaut)
- **D.** public, private, lazy
- **E.** public, private, mixed, protected

Suivant >

Question 2 de 41

Parmi les énoncés suivantes, laquelle est fausse en **Java** :

- **A.** Une classe peut implémenter une ou plusieurs interfaces
- **B.** Une classe abstraite peut contenir des méthodes non définies
- **C.** Une interface peut étendre une ou plusieurs interfaces
- **D.** Une interface peut contenir des méthodes non définies
- **E.** Une classe peut étendre une ou plusieurs classes

Quand est ce que ce code retourne true :

```
private static boolean process(boolean conditionA, boolean conditionB) {
    boolean result = false;
    if (conditionA) {
        if (conditionB) {
            return false;
        }
        result = true;
    }
    if (conditionB) {
        result = !result && conditionA;
    }
    return result;
}
```

- A. Quand conditionA est true et conditionB est true
- B. Quand conditionA est false et conditionB est false
- C. Quand conditionA est true et conditionB est false
- D. Quand conditionA est false et conditionB est true

Question 4 de 41

Que retourne le code suivant "List<Integer> years = new List<>();"

- A. Une liste avec un élément null
- B. Une liste vide
- C. Une erreur à l'exécution du code
- D. Une erreur à la compilation du code

Question 5 de 41

Parmi les éléments ci-dessous, lequel **n'est pas** un avantage de l'immuabilité (immutability en anglais) :

- A. facilite le multi-threading
- B. facilite la compréhension du flux d'exécution grâce à l'absence d'effets de bord
- C. facilite le caching
- D. accélère l'exécution

Question 6 de 41

Quel comportement a le code suivant :

```
class MyStringBuilder {
    public String stringA;
    public String stringB;

    public MyStringBuilder(String stringA, String stringB) {
        this.stringA = stringA;
        this.stringB = stringB;
    }

    public String concat() {
        return stringA + stringB;
    }
}
public class Main {
    private static void swapStrings(MyStringBuilder myStringBuilder) {
        String tmp = myStringBuilder.stringA;
        myStringBuilder.stringA = myStringBuilder.stringB;
        myStringBuilder.stringB = tmp;
    }
    public static void main(String[] args) {
        MyStringBuilder myStringBuilder = new MyStringBuilder("AA", "BB");
        swapStrings(myStringBuilder);
        System.out.println(myStringBuilder.concat());
    }
```

```
}
```

- A. Affiche :

  AAAA

- B. Affiche :

  BBBB

- C. Affiche :

  AABB

- D. Affiche :

  BBAA

- E. Déclenche une erreur de compilation

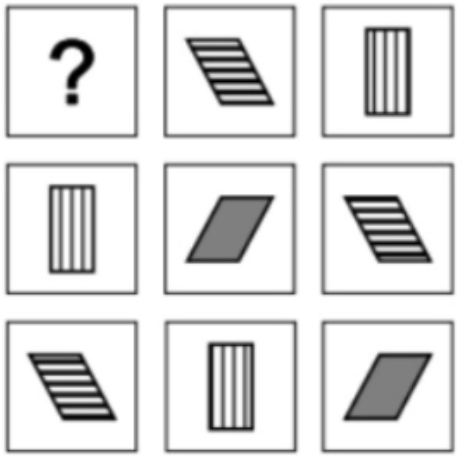
Quel énoncé est vraie par rapport au code ci-dessous (avec n >=0) :

```
public static int compute1(int n)
{
    int result = 0;
    for(int i=0; i<2; i++) {
        for(int j = 0; j < n; j++) {
            result++;
        }
    }
    return result;
}
```

```java
public static int compute2(int n)
{
    int result = 0;
    for(int i = 0; i < n; i++) {
        result++;
    }
    for(int i = 0; i < n; i++) {
        result++;
    }
    return result;
}
```

- A. compute1 et compute2 ne donnent pas le même résultat
- B. compute1 a une meilleure complexité temporelle que compute2
- C. compute1 et compute2 ont la même complexité temporelle
- D. compute2 a une meilleure complexité temporelle que compute1

Choisissez la bonne forme à mettre dans "?" :



- A. A
- B. B

Quel comportement a le code suivant :

```
class BaseClass {
    public String getMessage() {
        return "Hello from BaseClass";
    }
}
class DerivedClass extends BaseClass {
    @Override
    public String getMessage() {
        return "Hello from DerivedClass";
    }
}
public class Main {
    public static void main(String[] args) {
        BaseClass baseClass = new BaseClass();
        System.out.println(baseClass.getMessage());
        DerivedClass derivedClass = baseClass;
        System.out.println(derivedClass.getMessage());
    }
}
```

- A. Déclenche une erreur de compilation
- B. Déclenche une **NullPointerException** durant l'exécution
- C. Affiche :

  Hello from BaseClass

  Hello from DerivedClass
- D. Affiche :

  Hello from DerivedClass

  Hello from DerivedClass
- E. Affiche :

  Hello from BaseClass

  Hello from BaseClass

Question 10 de 41

Parmi les structures de données suivantes, laquelle offre la meilleure complexité temporelle pour la recherche ? (avec un nombre conséquent d'éléments)

- A. Arbre binaire de recherche
- B. Liste chainée
- C. Tableau trié
- D. Table de hashage

Question 11 de 41

À quelle fréquence les nouvelles versions majeures de la JDK sont publiées ?

- A. Tous les 24 mois
- B. Tous les 12 mois
- C. Tous les 6 mois
- D. Tous les 18 mois
- E. Tous les 3 mois

Question 12 de 41

Quelle est la complexité du code suivant :

```
public static void sort(int array[])
{
    int n = array.length;
    for (int i = 0; i < n-1; i++)
        for (int j = i+1; j < n; j++)
            if (array[i] > array[j])
            {
                // swap array[i] and array[j]
                int temp = array[j];
                array[j] = array[i];
                array[i] = temp;
            }
}
```

- A. O(logN)
- B. O(NlogN)
- C. O(N^2)
- D. O(N)

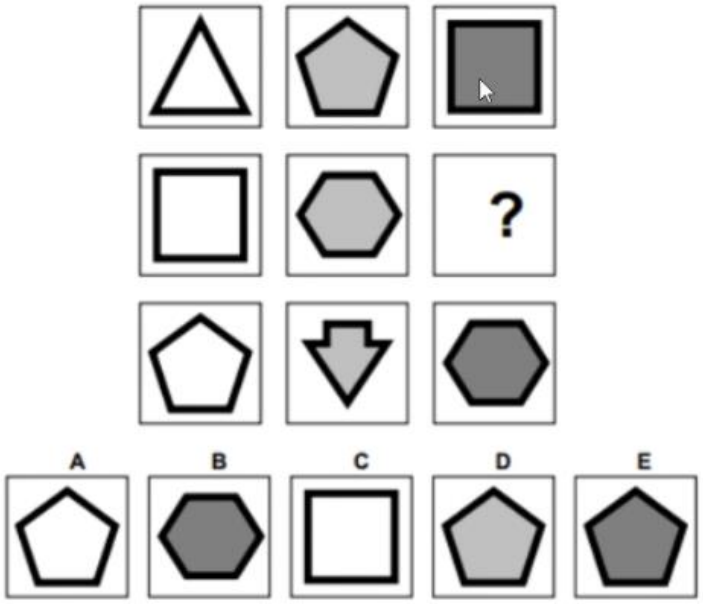
```java
import java.util.*;

public class HashSet
{
    public static void main(String args[])
    {
        Integer[] A = {22, 45,33, 66, 55};
        Integer[] B = {2, 83, 45, 12, 55};
        Set<Integer> set1 = new HashSet<Integer>();
        set1.addAll(Arrays.asList(A));
        Set<Integer> set2 = new HashSet<Integer>();
        set2.addAll(Arrays.asList(B));

        set1.addAll(set2);
        System.out.println(set1);
    }
}
```

- A. 2, 83, 45, 12, 55
- B. 33, 66, 2, 83, 22, 55, 12, 45
- C. 22, 45,33, 66, 55, 2, 83, 45, 12, 55
- D. 22, 45,33, 66, 55

Choisissez la bonne forme à mettre dans "?" :



- **A.** A
- **B.** B
- **C.** C
- **D.** D

Question 15 de 41

Une machine produit 100 unités de produit par minute. Si chaque caisse d'emballage contient 24 unités de ce produit, combien de caisses la machine remplit-elle en une heure ?

- **A.** 2,500
- **B.** 6,000
- **C.** 250
- **D.** 500
- **E.** 125

Question 16 de 41

L'encapsulation est un des concepts clés de la programmation orientée objet. Elle : (2 réponses à cocher)

- [ ] **A.** accélère la vitesse d'exécution du code
- [ ] **B.** permet d'hériter de plusieurs classes
- [x] **C.** permet de cacher les détails d'implémentation d'une classe
- [x] **D.** permet de garantir la cohérence et la validité des données d'une classe
- [ ] **E.** permet de réduire le nombre de lignes de code

Question 17 de 41

Sachant que l'attribut **list** est déclaré **final**, quel comportement a le code suivant :

```
import java.util.ArrayList;
import java.util.List;

class StringList {
    private final List<String> list = new ArrayList<>();

    public void addString(String value) {
        list.add(value);
    }
    public int size() {
        return list.size();
    }
}
public class Main {
    public static void main(String[] args) {
        StringList stringList = new StringList();
        stringList.addString("hello");
        System.out.println(stringList.size());
```

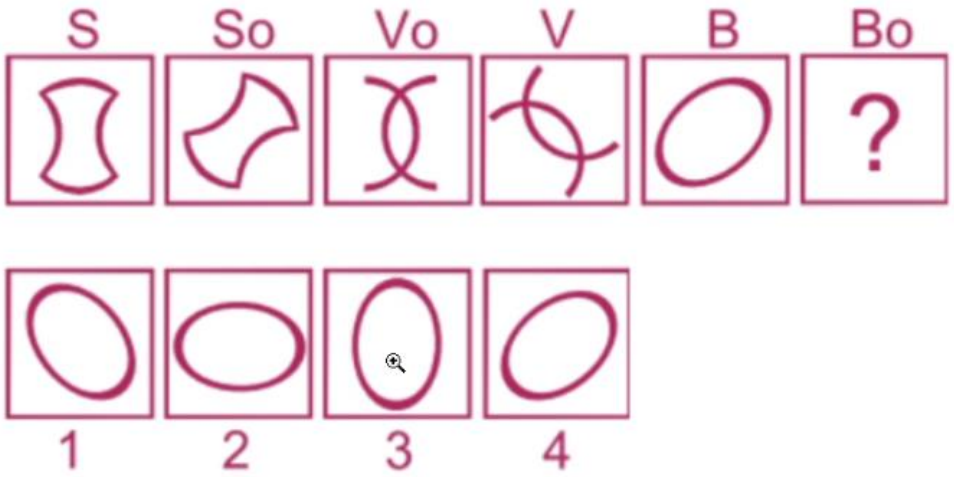
- **A.** Affiche :

  null
- **B.** Affiche :

  1
- **C.** Affiche :

  hello
- **D.** Déclenche une **FinalAttributeException** durant l'exécution
- **E.** Déclenche une erreur de compilation car l'attribut est **final**

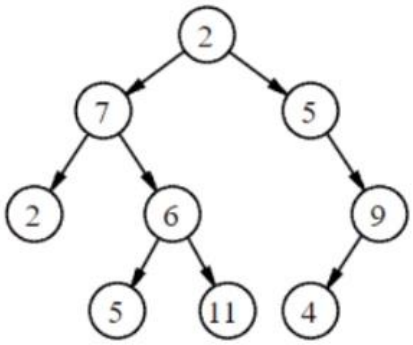
Choisissez la bonne forme à mettre dans "?" :



- **A.** 1
- **B.** 2
- **C.** 3
- **D.** 4

Question 19 de 41

Soit **Tree** une classe qui représente un arbre binaire. Que va afficher le code ci-dessous pour l'arbre donné en exemple s'il est appelé sur sa racine ?




```
public static void printTree(Tree tree) {
    if (tree.getLeft() != null)
        printTree(tree.getLeft());
    if (tree.getRight() != null)
        printTree(tree.getRight());
    System.out.println(tree.getValue());
}
```

- A. 5, 11, 4, 2, 6, 9, 7, 5, 2
- B. 2, 5, 11, 6, 7, 4, 9, 5, 2
- C. 7, 5, 2, 2, 6, 5, 11, 9, 4
- D. 2, 7, 2, 6, 5, 11, 5, 9, 4

Quelle est la complexité du code suivant :

```
public static boolean hasDuplicates(List<Integer> ints)
{
    for (Integer i: ints) {
        List<Integer> intsWithSameValue = ints.stream()
            .filter(element -> element.equals(i))
            .collect(Collectors.toList());
        if (intsWithSameValue.size() > 1) {
            return true;
        }
    }
    return false;
}
```

- A. O(logN)
- B. O(NlogN)
- C. O(N^2)
- D. O(N)

Qu'affiche le code suivant :

```
import java.util.HashMap;

public class Main {
    public static void main(String[] args) {
        // Create a HashMap object called capitalCities
        HashMap<String, String> flight = new HashMap<String, String>();

        // Add keys and values (Country, City)
        flight.put("Paris", "London");
        flight.put("London", "Berlin");
        System.out.println(flight.get("Berlin"));
    }
}
```

- A. Berlin
- B. Paris
- C. London
- D. null

Question 22 de 41

Qu'affiche ce code :

```
class Program
{
    static int process(int n, int p)
    {
        if (p == 1) {
            return n;
        }
        int result = process(n, p / 2);
        result *= result;
        if (p % 2 == 1) {
            result *= n;
        }
        return result;
    }

    public static void main (String args[])
    {
        System.out.println(process(3, 3));
    }
}
```

A. 12

B. 27

C. 24

D. 6

Question 23 de 41

Quel est le nombre de comparaison nécessaire dans un tableau d'entiers distints et non triés pour retrouver un élément qui n'est ni le max ni le min de ce tableau

- A. O(logN)
- B. O(1)
- C. O(NlogN)
- D. O(N)

Question 24 de 41

Qu'affiche le code javascript suivant :

```
for(var i = 0; i < 3; i++){
    function log(){
        console.log(i)
    }
    setTimeout(log, 100)
}
```

- A. 3 3 3
- B. 2 2 2
- C. 0 1 2
- D. Random combination of integers between 0 and 3

Question 25 de 41

Quelle ligne dans le code suivant pose problème le plus en terme de performance :

```
1  public static List<Instant> getDatesBeforeCovidLockdown
2          (List<Instant> dates) {
3      final String START_DATE = "2020-03-20T17:00:00.00Z";
4      List<Instant> result = new ArrayList<>();
5      for (Instant date: dates) {
6          Instant lockdownDate = Instant.parse(START_DATE);
7          if (date.isBefore(lockdownDate)) {
8              result.add(date);
9          }
10     }
11     return result;
12 }
```

- A. 4
- B. 7
- C. 6
- D. 3
- E. 8

Question 26 de 41

Sachant que l'attribut **list** est déclaré **final**, quel comportement a le code suivant :

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class StringList {
    private final List<String> list = Arrays.asList("hello");

    public void clear() {
        list = new ArrayList<>();
    }
    public int size() {
        return list.size();
    }
}
public class Main {
    public static void main(String[] args) {
        StringList stringList = new StringList();
        stringList.clear();
        System.out.println(stringList.size());
    }
}
```

- **A.** Affiche :

  0

- **B.** Déclenche une **FinalAttributeException** durant l'exécution

- **C.** Affiche :

  1

- **D.** Affiche :

  null

- **E.** Déclenche une erreur de compilation car l'attribut est **final**

Question 27 de 41

jwt est un token qui (2 réponses à cocher) :

- [ ] **A.** détermine le type de protocol à utiliser pour communiquer avec le serveur
- [x] **B.** peut encapsuler des informations concernant l'utilisateur connecté
- [x] **C.** contient une signature qui garantie l'identité de son créateur
- [ ] **D.** est utilisé sur les sites PWA pour gérer le mode offline
- [ ] **E.** permet de crytper les échanges entre le serveur et le navigateur

Question 28 de 41

Parmi les énoncés suivantes, laquelle est vraie en **Java** :

- ( ) **A.** Aucune des réponses
- (•) **B.** Si deux objets sont égaux, leurs **hashcodes** doivent être égaux
- ( ) **C.** Il n'y a aucune relation entre les hashcodes des objets et leur égalité
- ( ) **D.** Si deux objets ont le même **hashcode**, ils doivent être égaux

Question 29 de 41

En 2001, une société a vendu 730 000 unités de son produit. En 2001, son volume annuel atteignait 50% de son volume en 2004. Combien d'unités le volume de 2004 représente-t-il pour chacun des 365 jours de 2004 ?

- **A.** 5 000
- **B.** 1 100
- **C.** 4 000
- **D.** 1 000
- **E.** 2 000

Question 30 de 41

Quel comportement a le code suivant :

```
class BaseClass {
    public String getMessage() {
        return "Hello from BaseClass";
    }
}
class DerivedClass extends BaseClass {
    @Override
    public String getMessage() {
        return "Hello from DerivedClass";
    }
}
public class Main {
    public static void main(String[] args) {
        DerivedClass derivedClass = new DerivedClass();
        System.out.println(derivedClass.getMessage());
        BaseClass baseClass = derivedClass;
        System.out.println(baseClass.getMessage());
    }
}
```

- ○ **A.** Déclenche une erreur de compilation
- ○ **B.** Affiche :

  Hello from BaseClass

  Hello from BaseClass
- ○ **C.** Affiche :

  Hello from DerivedClass

  Hello from BaseClass
- ○ **D.** Déclenche une **NullPointerException** durant l'exécution
- ◉ **E.** Affiche :

  Hello from DerivedClass

  Hello from DerivedClass

Question 31 de 41

Parmi les énoncés suivantes, lesquelles sont vraies ? (2 réponses à cocher)

- [ ] **A.** Une méthode statique ne peut pas être **private**
- [ ] **B.** Une méthode statique peut être redéfinie (override) dans les classes filles
- [ ] **C.** Une méthode statique doit avoir au moins un argument
- [x] **D.** Une méthode statique ne peut pas utiliser des membres de classe non statiques
- [x] **E.** Une méthode statique doit être appelé avec le nom de sa classe

Quêstion 32 de 41

Quelle énoncé est fausse concernant les images Docker ?

- (•) **A.** Une image Docker est une représentation statique d'une application
- ( ) **B.** Une image Docker peut être publiée sur une registry Docker
- ( ) **C.** Une image Docker est une succession de couches superposées
- ( ) **D.** Une image Docker ne peut être exécuté qu'une seule fois en même temps sur une même machine

Que représente la classe X :

```
class X {
    private int value;
    private X next;
    private X previous;
}
```

- A. Un tableau dynamique
- B. Une liste simplement chainée
- C. Une liste doublement chainée
- D. Une table de hashage
- E. Un arbre binaire

Question 34 de 41

Quel comportement a le code suivant :

```
class ConsoleLogger {
    public void info(int number) {
        System.out.println("Integer: " + number);
    }
    public void info(String message) {
        System.out.println("String: " + message);
    }
    public void info(Object object) {
        System.out.println("Object: " + object.toString());
    }
}
public class Main {
    public static void main(String[] args) {
        ConsoleLogger logger = new ConsoleLogger();
        logger.info(-1);
        logger.info("0");
        logger.info((Object) "this is a string");
    }
}
```

- **A.** Affiche :

  Integer: -1

  String: 0

  Object: this is a string

- **B.** Déclenche une **NullPointerException** durant l'exécution

- **C.** Affiche :

  Integer: -1

  Integer: 0

  Object: this is a string

- **D.** Affiche :

  Object: -1

  Integer: 0

  String: this is a string

- **E.** Déclenche une erreur de compilation

Question 35 de 41

Quel comportement a le code suivant :

```
class Animal {
    protected String getNamePhrase() {
        return "I don't have a name";
    }
    public void sayHello() {
        System.out.println("Hello there! " + getNamePhrase());
    }
}
class Dog extends Animal {
    @Override
    protected String getNamePhrase() {
        return "I'm Bobby";
    }
}
public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Animal();
        animal1.sayHello();
        Animal animal2 = new Dog();
        animal2.sayHello();
    }
}
```

A. Déclenche une erreur de compilation

B. Affiche :

Hello there! I don't have a name

Hello there! I don't have a name

C. Affiche :

Hello there! I don't have a name

Hello there! I'm Bobby

D. Déclenche une **NullPointerException** durant l'exécution

E. Affiche :

Hello there! I'm Bobby

Hello there! I'm Bobby

Comment appelle-t-on cet algorithme ?

```
public static int process(
    int[] array, int key) {
    int low = 0;
    int high = array.length - 1;

    while (low <= high) {
        int mid = low + ((high - low) / 2);
        if (array[mid] < key) {
            low = mid + 1;
        } else if (array[mid] > key) {
            high = mid - 1;
        } else if (array[mid] == key) {
```

```
                return mid;
            }
        }
        return -1;
    }
```

- A. n'a pas de nom
- B. convergence low-high
- C. calcul clé unique
- D. recherche dichotomique

Quel énoncé est vraie par rapport au code ci-dessous (avec n >=0) :

```
public static int power1(int x, int n) {
    if (n == 0) {
        return 1;
    }
    return x * power1(x, n - 1);
}
```

```java
public static int power2(int x, int n) {
    int result = 1;
    while (n > 0) {
        result *= x;
        n--;
    }
    return result;
}
```

- A. power1 a une meilleure complexité temporelle que power2
- B. power1 et power2 ne donnent pas le même résultat
- C. power2 a une meilleure complexité temporelle que power1
- D. power1 et power2 ont la même complexité temporelle

Quel comportement a le code suivant :

```java
public class Main {
    private static void swapStrings(String a, String b) {
        String tmp = a;
        a = b;
        b = tmp;
    }
    public static void main(String[] args) {
        String firstname = "Gosling";
        String lastname = "James";
        swapStrings(firstname, lastname);
        System.out.println(firstname + ' ' + lastname);
    }
}
```

- **A.** Affiche :

  Gosling James

- **B.** Affiche :

  James James

- **C.** Affiche :

  James Gosling

- **D.** Affiche :

  Gosling Gosling

- **E.** Déclenche une erreur de compilation

Question 39 de 41

Dans quel cas choisiriez vous d'utiliser une Stack comme structure de donnée (2 réponses à cocher)

- [x] **A.** Pour une complexité temporelle optimale en cas d'accès au premier élément ajouté à la Stack
- [x] **B.** Pour stocker l'ordre d'exécution du code dans un compilateur
- [ ] **C.** Pour une complexité temporelle optimale en cas d'accès au dernier élément ajouté à la Stack
- [ ] **D.** Comme alternative à une hash map pour un sytème clef valeur
- [ ] **E.** Pour représenter une file d'attente

Question 40 de 41

Quel comportement a le code suivant :

```
class Ship {
    private static int totalSailedMiles = 0;

    public void sail(int miles) {
        totalSailedMiles += miles;
    }
    public int getTotalSailedMiles() {
        return totalSailedMiles;
    }
}
public class Main {
    public static void main(String[] args) {
        Ship shipA = new Ship();
        Ship shipB = new Ship();
        shipB.sail(10);
        Ship shipC = new Ship();
        shipC.sail(100);
        System.out.println(shipA.getTotalSailedMiles());
    }
}
```

- A. Affiche :

  0

- B. Affiche :

  110

- C. Déclenche une erreur de compilation car la méthode **getTotalSailedMiles** n'est pas statique

- D. Déclenche une erreur de compilation car **totalSailedMiles** est initialisé avec 0

Question 41 de 41

Un objet de type **Integer** en java occupe 128 bits en mémoire. Quel espace en mémore occupe l'objet list suivant : "List<Integer> list = new ArrayList<>(); list.add(1);"

- **A.** 128 bits
- **B.** 1 bit
- **C.** Plus de 1280 bits
- **D.** 0 bit