Faculté des Sciences
Aix*Marseille Université

Année universitaire 2021-2022

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 1h30
Examen de : L2 Nom du diplôme : Licence d'informatique
Code du module : SIN3U02 Libellé du module : Programmation 2
Calculatrices autorisées : NON Documents autorisés : OUI

---

1 Consignes

Inscrivez votre nom et prénom ci-dessous :
Nom :

Prénom :

Répondez directement sur le sujet en cochant ou en écrivant vos réponses aux emplacements prévus à cet effet.

2 Questions de cours

2.1 Question sur les exceptions

On considère le code suivant :

```
public class ListException {
  public static String get(List<String> strings, int index){
    try {
      String result = strings.get(index);
      if(result == null)
        return "null";
      else
        return result;
    }
    catch (IndexOutOfBoundsException e){
      return "OE";
    }
    catch (NullPointerException e){
      return "NE";
    }
  }

  public static void main(String[] args){
    List<String> strings = new ArrayList<>();
    strings.add("toto");
    strings.add(null);
    strings.add("tata");
    for(int index = 0; index < 4; index++){
      System.out.print(get(null, index) + " ");
      System.out.print(get(strings, index) + " ");
    }
  }
}
```

Quel est l'affichage qui est produit par l'exécution de la méthode `main` de ce code ? Cochez la bonne réponse.
☐ NE NE NE NE NE NE NE NE
☒ NE toto NE null NE tata NE OE
☐ NE toto NE null NE tata NE NE
☐ null toto null null null tata null null
☐ pas d'affichage car le code ne compile pas
☐ pas d'affichage car l'exécution de ce code lève une exception

2.2 Question sur les types

Pour chaque ligne du tableau ci-dessous cochez le ou les qualificatifs qui sont vrais pour le type de la colonne de gauche :

type	classe	interface	type primitif	type paramétré
`ArrayList<T>`	☒	☐	☐	☒
`Double`	☒	☐	☐	☐
`Iterable<T>`	☐	☒	☐	☒
`double`	☐	☐	☒	☐

2.3 Question sur les définitions

Que peut-on définir à l'intérieur de la déclaration d'une interface, d'une énumération (de type `enum`), d'une classe abstraite ou d'une classe concrète (classe n'étant pas abstraite) ? Pour chaque ligne du tableau ci-dessous cochez chaque type d'éléments (attribut d'instance, attribut de classe `static`, constructeur, méthode d'instance avec code, méthode de classe `static` avec code) qui peut être défini à l'intérieur de la déclaration d'un type de la colonne de gauche :

type	attribut d'instance	attribut de classe	constructeur	méthode d'instance	méthode de classe
interface	☐	☐/☒	☐	☐/☒	☐/☒
enum	☒	☒	☒	☒	☒
classe abstraite	☒	☒	☒	☒	☒
classe concrète	☒	☒	☒	☒	☒

2.4 Question sur l'égalité

On considère trois chaînes de caractères (`string1`, `string2` et `string3`) instanciées puis modifiées par le code suivant :

```
String string1 = new String("toto");
String string2 = new String("toto");
String string3 = new String("tata");
```

Quelle est la valeur de `string1 == string2` ? Répondez directement en dessous.

`false`

Quelle est la valeur de `string1 == string3` ? Répondez directement en dessous.

`false`

Quelle est la valeur de `string1.equals(string2)` ? Répondez directement en dessous.

`true`

Quelle est la valeur de `string1.equals(string3)` ? Répondez directement en dessous.

`false`

Quelle est la valeur de `string2.equals(string3)` ? Répondez directement en dessous.

`false`

2.5 Question sur les types paramétrés

Dans une classe `MyClass<T>`, quel type d'attribut est valide en Java ? On suppose qu'on a importé `List` dans la classe. Cochez la ou les bonnes réponses.
☐ `E`
☒ `T`
☐ `T<List<T>>`
☒ `List<T[]>`
☒ `T[][]`
☒ `List<List<T>>`
☒ `List<T>[]`
☐ `List<E>`

2.6 Question sur les tableaux

On considère le code suivant :

```
public class AppMatrixStrings {
  public static void main(String[] args){
    String[][] matrixOfStrings = new String[3][];

    for(int row = 0; row < matrixOfStrings.length; row++) {
      for (int column = 0; column < matrixOfStrings[row].length; column++)
        matrixOfStrings[row][column] = "(" + row + ", " + column + ")";
    }

    for(String[] line : matrixOfStrings)
      for(String value : line)
        System.out.print(value);
  }
}
```

Quel est l'affichage qui est produit par ce code ? Cochez la bonne réponse.
☐ (0, 0)(1, 0)(1, 1)(2, 0)(2, 1)(2, 2)
☐ (0, 0)(0, 1)(0, 2)(1, 0)(1, 1)(1, 2)(2, 0)(2, 1)(2, 2)
☐ (1, 0)(2, 0)(2, 1)
☐ (1, 1)(1, 2)(1, 3)(2, 1)(2, 2)(2, 3)(3, 1)(3, 2)(3, 3)
☐ pas d'affichage car le code ne compile pas
☒ pas d'affichage car l'exécution de ce code lève une exception

3 Extension de classe

On considère deux classes `Rectangle` et `ColoredRectangle` définies par le code suivant :

```
public class Rectangle {
  final int x;
  final int y;
  final int width;
  final int height;

  public Rectangle(int x, int y, int width, int height) {
```

```
    this.x = x;
    this.y = y;
    this.height = height;
    this.width = width;
  }
}

public class ColoredRectangle extends Rectangle {
    Color color;

    public ColoredRectangle(int x, int y, int width, int height, Color color) {
        /* code manquant ? */
        this.color = color;
    }
}
```

3.1 Question code manquant

Quelle est la partie de code qui manque dans la première ligne du constructeur de `ColoredRectangle` (à la place du commentaire) ? Cochez la bonne réponse.

☐ `this()`
☐ `this(x, y, width, height)`
☐ `super()`
☒ `super(x, y, width, height)`
☐ Il ne manque pas de code !

3.2 Question sur les types (1/2)

Quels sont les affectations qui ne produisent pas d'erreurs à la compilation ? Cochez la ou les bonnes réponses.

☒ `Rectangle rectangle = new Rectangle(1, 1, 100, 100)`
☒ `Rectangle rectangle = new ColoredRectangle(0, 0, 100, 100, Color.BLACK)`
☐ `Rectangle rectangle = new Object()`
☐ `ColoredRectangle coloredRectangle = new Rectangle(1, 1, 10, 10)`
☐ `ColoredRectangle coloredRectangle = new Object()`
☒ `Object object = new Rectangle(0, 0, 10, 10)`
☒ `Object object = new ColoredRectangle(0, 0, 10, 10, Color.BLACK)`

3.3 Question sur les types (2/2)

On considère trois listes déclarées et initialisées par le code suivant :

```
List<ColoredRectangle> coloredRectangles = new ArrayList<>();
List<Rectangle> rectangles = new ArrayList<>();
List<Object> objects = new ArrayList<>();
```

En sachant que la méthode `add` de `List<E>` a pour signature `boolean add(E e)`. Quels sont les appels de `add` qui ne produisent pas d'erreurs à la compilation ? Cochez la ou les bonnes réponses.

☒ `rectangles.add(new Rectangle(10, 10, 100, 100))`
☒ `rectangles.add(new ColoredRectangle(0, 0, 10, 10, Color.BLACK))`
☐ `rectangles.add(new Object())`
☒ `objects.add(new Object())`
☐ `coloredRectangles.add(new Rectangle(0, 0, 10, 10))`
☐ `coloredRectangles.add(new Object())`
☒ `objects.add(new Rectangle(20, 20, 20, 20))`
☐ `objects.add(new Rectangle(20, 20))`
☐ `objects.add(new ColoredRectangle(0, 0, 10, 10))`

FIGURE 1 – Damier 10×10

4 Damier

Pour cet exercice, on vous demandera de donner le code de certaines classes avec un certains nombres d'éléments : méthodes et attributs demandées. Vous avez (et devez dans certains cas) ajouter de vous-même des attributs ou méthodes qui vous semble nécessaires. On ne vous donnera pas les modificateurs d'accès pour les différents éléments du code. Ce sera donc à vous de choisir l'accessibilité des éléments entre : `private`, `public`, `protected` et *default* (pas de mot-clé).
Pour cet exercice, on va s'intéresser au jeu de dames et à la modélisation des différents éléments d'un tel jeu : pion (*pawn*), couleur (*color*), case (*square*) et damier (*game board*). Il existe deux variantes du jeu de dames : les dames françaises où le plateau a une taille de 10×10 cases et les dames anglaises (ou checkers) où il a une taille 8×8. D'autres tailles de plateau sont envisageables comme une variante en 12×12 mais la taille doit toujours être un entier strictement positif qui est pair. La figure ci-dessus présente un plateau de taille 10×10.
Le type `color` permet de représenter la couleur d'une case. Un objet de type `Color` correspond à une couleur blanche ou noire. On souhaite disposer pour le type `Color` d'une méthode `otherColor` qui ne prend pas d'argument et qui renvoie l'autre couleur que celle pour laquelle elle a été appelée (pour blanc, elle renvoie noir, et pour noir elle renvoie blanc).

— **Questions 1 :** *Donnez le code java du type* `Color`

```
public enum Color {
  WHITE, BLACK;

  public Color otherColor() {
    if (this == WHITE)
      return BLACK;
    else
      return WHITE;
  }
}
```

Un objet de la classe `Square` permet de représenter une case d'un damier. Cette classe possède les éléments suivants :

— un attribut `color` de type `Color` ;
— un attribut `pawn` de type `Pawn` (on supposera que la classe `Pawn` permettant de représenter un pion est déjà définie), cet attribut vaudra `null` si aucun pion n'est présent dans la case ;
— un constructeur prenant comme argument une `color` de type `Color`, construisant une case de la couleur demandée sans pion ;
— une méthode `getColor` qui n'a pas d'argument et renvoie une couleur de type `Color` ;
— une méthode `getPawn` qui n'a pas d'argument et renvoie une pion de type `Pawn` ;
— une méthode `setPawn` qui a un argument de type `Pawn` et ne renvoie rien.
— **Questions 2 :** *Donnez le code java du type* ***`Square`***

```java
public class Square {
  private Color color;
  private Pawn pawn;

  public Square(Color color) {
    this.color = color;
  }

  public Pawn getPawn() {
    return this.pawn;
  }

  public void setPawn(Pawn pawn) {
    this.pawn = pawn;
  }

  public Color getColor() {
    return this.color;
  }
}
```

Les cases d'un damier sont mémorisées sous la forme d'un tableau à deux dimensions d'objets de type `Square`. La première case, d'indice à la ligne `0` et colonne `0`, est toujours noire, puis les couleurs des cases suivantes alternent entre blanc et noir, comme sur la figure (la première case est celle en haut à gauche). Un objet de la classe `GameBoard` permet de représenter un damier. Cette classe possède les éléments suivants :

— des attributs à déterminer ;
— un constructeur qui prend en paramètre la taille du damier, ce constructeur devra lever une message `IllegalArgumentException` avec un message explicatif si la taille est inférieure ou égale à 0 ou bien n'est pas paire ;
— une méthode `getSize` qui n'a pas d'argument et renvoie la taille du damier ;
— une méthode `getSquare` qui prend deux argument : un numéro de ligne et numéro de colonne et qui renvoie la case de la ligne et colonne correspondantes. Cette méthode devra lever une exception de type `IndexOutOfBoundsException` avec un message explicatif si les indices ne sont pas valides (négatif ou plus grand que la taille).

On souhaite pouvoir utiliser la classe `GameBoard` de la façon suivante sans que cela provoque des erreurs.

```java
public class mainGame {
  public static void main(String[] args){
    GameBoard gameBoard = new GameBoard(10);
    gameBoard.getSquare(3,3).setPawn(new Pawn());
  }
}
```

Les classes `IllegalArgumentException` et `IndexOutOfBoundsException` étendent `RuntimeException` et ont chacune une constructeur prenant en argument une chaîne de caractères de type `String` correspondant au message de l'exception.

— **Questions 3 :** *Donnez le code java du type* ***`GameBoard`***

```
public class GameBoard {

  private final Square[][] squares;

  public GameBoard(int size) {
    if(size < 0){
      throw new IllegalArgumentException("A game board must have a positive size and not equal to "
        + size);
    }
    if(size%2 == 1){
      throw new IllegalArgumentException("A game board must have an even size and not equal to "
        + size);
    }
    this.squares = new Square[size][size];
    this.initSquares(size);
  }

  private void initSquares(int size) {
    Color color = Color.BLACK;
    for (int row = 0 ; row < size ; row ++) {
      for (int column = 0 ; column < size ; column ++) {
        this.squares[row][column] = new Square(color);
        color = color.otherColor();
      }
    }
  }

  public int getSize() {
    return this.squares.length;
  }

  public Square getSquare(int row, int column){
    requiresIndexInRange(row, "Row");
    requiresIndexInRange(column, "Column");
    return squares[row][column];
  }


  private void requiresIndexInRange(int index, String indexKind) {
    if(index <0 || index >= getSize()){
      throw new IndexOutOfBoundsException(indexKind + "index " + index
        + " out of bounds for size " + getSize());
    }
  }
}
```