l2211

By
Daniel
Dalati

2023

Faculté des Sciences
Section 3

Année : 2017 – 2018
P + F : S2

Cours : I 2 2 11
Durée : 2h : 30

مطلوب الاجابة على كل جزء بشكل متواصل ودون تداخل مع الجزء الاخر.

Partie P (Partiel) : (sur 100 points pour ~ 45 minutes)

Exercice 1 (60 points)

a) Implémenter la classe Complexe qui représente les nombres complexes (partie réel et partie imaginaire). La classe doit contenir :
- Deux attributs privés de type *double* (*reel*, *img*) ;
- Un premier constructeur qui initialise les attributs par les valeurs de paramètres ;
- Un autre constructeur qui initialise les attributs à 0 en faisant appel au premier ;
- Une méthode *toString* qui retourne le mot *reel* + *img i* ;
- Une méthode *additionner* qui retourne un objet Complexe représentant la somme de l'objet courant avec le paramètre. [Indication : $(a_1 + b_1 i) + (a_2 + b_2 i) = (a_1 + a_2) + (b_1 + b_2) i$] ;
- Une méthode *multiplier* qui retourne un objet Complexe représentant la multiplication de l'objet courant avec le paramètre. [Indication : $(a_1 + b_1 i) \times (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + a_2 b_1) i$].

b) Implémenter une méthode *main* dans laquelle vous calculez et affichez la valeur de i^2 : [(0 + 1 *i*) × (0 + 1 *i*) = (-1 + 0 *i*)].

Exercice 2 (40 points)

On considère la classe *Mois* suivante dont les 12 objets possibles sont déjà instanciés. On suppose que le mois de Février contient toujours 28 jours.

```
public final class Mois {

    public final static Mois JANVIER = new Mois("Janvier", 1, 31);
    public final static Mois FEVRIER = new Mois("Fevrier", 2, 28);
    public final static Mois MARS = new Mois("Mars", 3, 31);
    ...

    private final String nom;
    private final int rang;
    private final int nbJours;

    private static final Mois[] values = {JANVIER, FEVRIER, MARS, ...};

    public static Mois getMois(int rang){
        // à compléter
    }

    private Mois(String nom, int rang, int nbJours) {
        this.nom = nom;
        this.rang = rang;
        this.nbJours = nbJours;
    }
```

```
    public String getNom() {
        return nom;
    }

    public int getRang() {
        return rang;
    }

    public int getNbJours() {
        return nbJours;
    }

    public Mois suivant() {
       //à compléter
    }
}
```

a) Compléter la méthode statique *getMois* qui étant donné le rang du mois, retourne l'objet Mois correspondant (*getMois*(4) retourne *AVRIL*).

b) Compléter la méthode *suivant* qui retourne le mois qui suit l'objet *Mois* courant. (Noter que si l'objet courant est *DECEMBRE*, le mois suivant doit être *JANVIER*.)

On considère maintenant la classe Date suivante :

```
public class Date {
    private int jour;
    private Mois mois;
    private int annee;

    public Date(int jour, Mois mois, int annee) {
        this.jour = jour;
        this.mois = mois;
        this.annee = annee;
    }

    public Date(int jour, int mois, int annee) {
        //à compléter
    }

    public Date suivant(){
        //à compléter
    }
}
```

c) Compléter le constructeur ainsi que la méthode *suivant* dans la classe *Date*. (La méthode *suivant* retourne l'objet *Date* correspondant au jour qui suit la date courante.)

(Final) : (sur 100 points pour ~ 105 minutes)

Exercice 3 (30 points)

On considère les classes *Date* et *Mois* de l'exercice précédent (on peut supposer que toutes les méthodes sont implémentées).

a) Redéfinir dans la classe *Date* la méthode *equals* héritée de la classe Object (deux dates sont égales si tous les attributs sont égaux).

b) Changer l'implémentation de la classe *Date* afin que les objets de type *Date* soient comparables entre eux (on peut comparer les objets Mois selon leur rang).

c) Réécrire le premier constructeur de la classe *Date* de façon à ce qu'il lance une exception *DateNonValide* (que vous deviez aussi implémenter) lorsque le paramètre *jour* est négatif ou lorsqu'il dépasse le nombre de jours du paramètre *mois*. Montrer l'utilisation de ce constructeur dans une méthode *main*.

```
public interface Comparable<T>{
    public int compareTo(T o) ;
}
```

Exercice 4 (35 points)

On suppose que l'on veut implémenter un jeu simple qui consiste en des objets graphiques (Cercles, Rectangles, Triangles, …) qui se déplacent aléatoirement sur un Canvas et qui ont de l'énergie. Le joueur doit cliquer sur chaque objet plusieurs fois (5 fois) afin qu'il disparait. Le jeu se termine lorsque tous les objets auront disparu.

Pour ce faire, on procède comme suit :

```
public class Cercle {
    private double x, y;
    private double rayon;
    private int energie = 5;

    public Cercle(double x, double y, double rayon) {
        this.x = x;
        this.y = y;
        this.rayon = rayon;
    }

    public void deplacer(double dx, double dy){
        x += dx;
        y += dy;
    }

    public boolean contient(double x, double y){
        //retourne vrai si la distance entre le centre et le point passe
        //en parametre est < rayon càd le point (x,y) est dans le cercle
        double dx = this.x - x;
        double dy = this.y - y;
        return Math.sqrt(dx*dx + dy*dy) < rayon;
}
```

```
    public void reduireEnergie(){
        if(energie > 0){
            energie --;
        }
    }

    public int getEnergie() {
        return energie;
    }
}

public class Rectangle {
    private double x, y;
    private double longueur, largeur;
    private int energie = 5;

    public Rectangle(double x, double y, double longueur, double largeur)
{
        this.x = x;
        this.y = y;
        this.longueur = longueur;
        this.largeur = largeur;
    }

    public void deplacer(double dx, double dy){
        x += dx;
        y += dy;
    }

    public boolean contient(double x, double y){
        //retourne vrai si le point passe en parametre est dans le
rectangle
        double dx = x - this.x;
        double dy = y - this.y;
        return dx >=0 && dy >= 0 && dx <= largeur && dy <= longueur;
    }

    public void reduireEnergie(){
        if(energie > 0){
            energie --;
        }
    }

    public int getEnergie() {
        return energie;
    }
}
```

```
Public class Canvas {// classe simplifiée
    private Cercle[] ensCercle = new Cercle[100];
    private int nbCercle = 0;
    private Rectangle[] ensRectangle = new Rectangle[100];
    private int nbRect = 0;

    public void ajouterCercle(Cercle c){
        ensCercle[nbCercle++] = c;
    }
    //...

    private void eliminerCercle(Cercle c){
        //chercher c, decaler les elements du tableau a gauche
    }

    //Pareil pour Rectangle

    public void attaquer(double x, double y){
        for(int i =0; i < nbCercle; i++){
            if(ensCercle[i].contient(x, y)){
                ensCercle[i].reduireEnergie();

            }
            if(ensCercle[i].getEnergie() == 0){
                eliminerCercle(ensCercle[i]);

            }
        }

        // Pareil pour Rectangle
    }
```

Vous remarquez que dans la classe *Canvas*, on doit créer un tableau pour chaque type d'objets graphique, ainsi que les méthodes *ajouter* et *eliminer*. Dans la méthode *attaquer*, il faut implémenter des boucles pour chaque type d'objet aussi… Si l'on veut ultérieurement utiliser autres types d'objets graphiques, il faut changer le code de la classe *Canvas* en ajoutant tout le traitement pour les nouveaux types d'objets.

a) Faire une généralisation des classes *Cercle* et *Rectangle* par une classe *Forme*.

b) Réécrire la classe Cercle.

c) Réécrire la classe *Canvas* en utilisant la classe *Forme* et en utilisant un *ArrayList* au lieu des tableau

```
public class ArrayList<T>…{
    public void add(T e){…}
    public void remove(T e){…}
    public int size(){…}
    public T get(int i) {…}
}
```

Exercice 5 (35 points)

On considère la classe Banque suivante :

```
public class Banque {

    private class Compte {

        private String ref;
        private int solde;

        Compte(String ref) {
            this.ref = ref;
        }

        void crediter(int somme) {
            solde += somme;
        }

        void debiter(int somme) {
            solde -= somme;
        }
    }

//Map qui associe un compte à sa référence.
    private Map<String, Compte> mapCompte = new HashMap();

    public void creerCompte(String ref) {
        //à compléter
    }

    public void crediter(String ref, int somme) {
        // à compléter
    }
}
```

La classe Compte est une classe interne. Seulement à partir d'un objet Banque, on peut créer des comptes et les créditer ou les débiter.

a) Compléter la méthode *creerCompte* qui associe au String *ref*, un nouvel objet compte.

b) Compléter la méthode *crediter* qui devra récupérer l'objet compte associé à *ref* et le créditer de la somme. (Il faut tester si *ref* est une clé de la *Map*.)

En fait, une banque enregistre toujours les opérations sur un compte (un client peut toujours consulter les opérations de crédits ou de débits effectuées sur son compte). L'implémentation ci-dessus ne tient pas compte de ce fait.

Une Ecriture correspond à une opération sur un compte. On la représente par la classe suivante :

```
public class Ecriture {
    private String intitule ; //credit ou debit
    private int somme;

    public Ecriture(String intitule, int somme) {
        this.intitule = intitule;
        this.somme = somme;
    }
```

```
    public String getIntitule() {
        return intitule;
    }

    public int getSomme() {
        return somme;
    }
}
```

c) Ajouter à la classe Banque une Map qui associe la référence d'un compte à une liste d'écritures.
d) Changer l'implémentation de la méthode *creerCompte* afin qu'elle associe le nouveau compte crée à une liste d'écriture vide.
e) Changer l'implémentation de la méthode *crediter* afin qu'elle ajoute une nouvelle écriture dans la liste des écritures associées au compte en question.
f) Ajouter à la classe Banque la méthode *List<Compte> chercherComptes (int limite)* qui retourne la liste des comptes dont la valeur de leur dernière opération de crédit dépasse la valeur *limite*.

```
public interface Map<K,V> …{
    public void put(K key, V value) ;
    public boolean containsKey(K key) ;
    public V get(K key) ;
}
```

1 → List <compte>
2 → coll <compte> coll = mapcompte.values();

(I2211)

Partie pratique :

X 1) attributs de type privée

```
Public class Complex {
    private double reel, imag;
    public complexe (double reel, double imag) {
        this.reel = reel;
        this.imag = imag;
    }
    public Complexe () {
        this (0.0, 0.0);
    }
    public complexe additionner (complexe c) {
        Complexe c1 = new Complexe();
        c1.reel = this.reel + c.reel;
        c1.imag = this.imag + c.imag;
        return c1;
    }
public string toString() {
    return reel + " + " + imag + "i";
    }
public complexe multiplier (complexe c) {
    Complexe c2 = new complexe();
    c2.reel = this.reel * c.reel - this.imag * c.imag
    c2.imag = this.reel * c.imag + this.imag * c.reel
    return c2; }
Void main (-) {
    Complexe c1 = new Complexe (0,1);
    Complexe c2 = c1.multiplier (c1); Sout (c2); }
```

Ex2) a) . public static Mois getMois (int rang)
return values [rang - 1]; }

b) . public Mois suivant () {
if (this.rang == values.length) {
return janvier;
}
return values [this.rang];
}

c) . public Date (int jour, int mois, int annee)
{ this.jour = jour;
this.mois = Mois.getMois (mois);
this.annee = annee;
}

. public Date suivant () {
int j = jour;
Mois m = mois;
int a = annee;
if (j == mois.getNbJours()) {
j = 1;
m = mois.Suivant();
if (m == Mois.janvier) {
a++;
}
} else {
j++;
}
return new Date (j, m, a);
}

Partie final :

3) • méthode equals :

```
public Boolean equals (Object o) {
    if (!(o instanceof Date)) {
        return false;
    }
    Date d = (Date) o;
    return jour == d.jour && mois == d.mois &&
           annee = d.annee;
}
```

changer l'implémentation de la classe Date

• public class Date implements Comparable <Date> {

```
public int compareTo (Date d) {
    // transforme data en jour
    int d1 = annee * 365;
    for (int i=1; i<mois.getRang(); i++) {
        d1 = d1 + Mois.getMois(i).getNbJours();
    } d1 = d1 + jour;
    int d2 = d.année * 365;
    for (int i=1; i < mois.getRang(); i++) {
        d2 = d2 + Mois.getMois(i).getNbJours();
    }
    d2 = d2 + d.jour;
    return d1 - d2;
}
}
```

```
• public class DateNonValide extends Exception
    public DateNonValide (string message) {
        super (message);
    }
}
public Date (int jour, Mois mois, int année) throws
    DateNonValide {
  if (jour < 1 || jour > mois.getNbJours()) {
      throw new DateNonValide ("jour = " + jour);
  }
  this.jour = jour;
  this.mois = mois;
  this.annee = annee;
}
public static void main (String[] args) {
    try {
      Date d = new Date (23, Mois.janvier, 2000);
    } catch (DateNonValide ex) {
        // ---
    }
}
```

Ex 4) • Généralisation de classe : public abstract class -- CSMU

a) public abstract classe Forme {

```
    protected double x, y;
    protected int énergie = 5;
public forme (double x, double y) {
    this.x = x;
    this.y = y;
}
```

```
public void deplacer (double dx, double dy ) {
        //---  (ñ méthode)
}

public void reduireEnergie () {
        //---
}

public abstract boolean contient (double x, double y);
public int getEnergie () {
    return energie;
  }
}
```

b)

```
public class cercle extends Forme {
        private int rayon;
    public cercle (double x, double y , int rayon) {
        super(x, y))
        this.rayon = rayon;
    }
public Boolean contient (double x, double y) {
        //----
    }
    public int getRayon () {
        return rayon;
    }
}
```

En utilisant classe form

```
public class Canvas. {
    private List<Forme> ls = new ArrayList<>();
public void ajouter (Forme F) {
        ls.add(F);
}
```

Ex: ls.add(new etudiant("13", "jean"));

```
public void attaquer (double x, double y) {
    for (int i=0; i < ls.size(); i++) {
        if (ls.get(i).contient(x, y)) {
            ls.get(i).reduireEnergie();
        }
        if (ls.get(i).getEnergie() == 0) {
            ls.remove(ls.get(i));
        }
    }
}
```

EX 5) a) public void creerCompte (String ref) {
mapCompte.put(ref, new Compte(ref));
}

b) public void crediter (String ref, int somme) {
if (mapCompte.containsKey(ref)) {
mapCompte.get(ref).crediter(somme);
}
}

c) private Map<String, List<Ecriture>> mape = new HashM

d) public void creerCompte (String ref) {
Compte c = new Compte(ref);
mapCompte.put(ref, c);
mape.put(ref, new ArrayList<Ecriture>());
}

e) public void crediter (String ref, int somme) {
if (mapCompte.containsKey(ref)) {
mapCompte.get(ref).crediter(somme);
Ecriture e = new Ecriture ("credit", somme);
mape.get(ref).add(e); }

List<Compte> chercherComptes (int limite)

UNIVERSITE LIBANAISE
FACULTE DES SCIENCES
BRANCHE: 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours	**: I2211 = Info 302**	**English**	**Date : Juillet 2019**
Examen	**: Final**		**Durée : 2 heures**

Exercice 1 (40 points)

We consider the class Couple which represents the couples of integers (eg The couple (1,2)) as follows:

```
public class Couple {

    private int a, b;

    public Couple(int a, int b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public String toString() {
        return "(" + a + "," + b + ")";
    }
}
```

a) Add to the class the equals method inherited from the Object class that returns true if the attributes are equal one by one;
b) Add to class a static method *produitCartesien* which given two arrays of integers in parameters returns an array of couples representing the corresponding Cartesian product ({3,1} × {5,2,4} = {(3,5), (3,2), (3,4), (1,5), (1,2), (1,4)}) ;

In a main method, we have the following instructions:

```
int[] tab1 = {3,1} ;
int[] tab2 = {5,2,4} ;
```

c) Write the instruction to calculate the Cartesian product of *tab*1 and *tab*2 and store it in an array of couples.
d) Write the instructions for sorting the array of couples in ascending order according to the first attribute, and if the first attributes are equal, sort them according to the second attribute using the sort method of the Arrays class and a Comparator. (E.g. the above array becomes sorted as follows : {(1,2), (1,4), (1,5), (3,2), (3,4), (3,5)}.)

We want to prohibit any creation of couples with negative numbers.

e) Implement the NumberNegativeException class that inherits from the Exception class;
f) Re-implement the constructor to throw a NumberNegativeException if one of the parameters is negative with a message indicating the given parameter and the negative value.
g) In a main method, show how to use this constructor.

```
public interface Comparator<T>{
  public int compare (T o1, T o2);
}
```

Exercice 2 (30 points)

We consider the class *GestionPatient* that manages the customers of a pharmacy. This class defines a *HashMap* that associates the name (String) of a client with the list of drugs (List <String>) corresponding to its prescription. Implement the four methods following the instructions given in comments. (Use ArrayList and if needed the methods add, get and contains)

```
public class GestionPatient {
    private Map<String, List<String>> map = new HashMap();

    public void ajouterPatient(String p){
    // associates patient p (if it does not exist) to an empty list.
    }

    public void ajouterMed(String p, String med){
    // add the drug med to the list corresponding to the patient p.
    }

    public List<String> getOrdonnance(String p){
    // return the list of the patient's medication p.
    }

    public List<String> getPatients(String med){
    // returns a list of all patients whose prescription
    //contains the drug med.
    }
}
```

Exemple d'exécution :

```
    GestionPatient gp = new GestionPatient();
    gp.ajouterPatient("jean"); gp.ajouterPatient("jack");
    gp.ajouterPatient("olivier"); gp.ajouterMed("jean", "augmentin");
    gp.ajouterMed("jean", "panadol"); gp.ajouterMed("jean", "doliprane");
    gp.ajouterMed("jack", "nexium"); gp.ajouterMed("jack", "augmentin");
    gp.ajouterMed("olivier", "nexium");gp.ajouterMed("olivier", "augmentin");
    System.out.println(gp.getPatients("augmentin"));
Résultat :  run:
      [olivier, jean, jack]
```

```
public interface Map <K, V> {
  public void put(K key, V value);
  public V get(K key);
  public boolean containsKey(K key);
  public Set<K> keySet();
  public Set<Map.Entry<K, V>> entrySet();
  public int size();

  interface Entry<K,V> {
    K getKey();
    V getValue();
  }
}
```

Exercice 3 (30 points)

In a university, employees are classified in two categories: Staff and Instructor. Staff and instructors have a fixed salary throughout the year. However, instructors may have an overload, which is paid according to a formula applied to their annual salary. All employees can have their salary increased by a certain percentage. We thus consider the two following classes:

```
public class Staff {

    private String name;
    private double salary;

    public Staff(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public void raiseSalary(int percentage) {
        salary += salary * percentage / 100;
    }

    public double getSalary() {
        return salary;
    }
}

public class Instructor {

    private String name;
    private double salary;
    private int overload;

    public Instructor(String name, double salary, int overload) {
        this.name = name;
        this.salary = salary;
        this.overload = overload;
    }

    public void raiseSalary(int percentage) {
        salary += salary * percentage / 100;
    }

    public double getSalary() {
        return salary + salary * 12 / 315 * overload;
    }
}
```

The application requires a collection (or table) in which all employees are stored in order to calculate the sum of all salaries due to employees.

To do this, we need to construct an abstraction of these two classes into an Employe class from which the created objects can execute all their methods.

Implement the class Employe and rewrite the class Instructor.

UNIVERSITE LIBANAISE
FACULTE DES SCIENCES
BRANCHE: 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours	**: I2211**		**Date : Oct, 2020**
Examen	**: 2ème Session**	**English**	**Durée : 1 :15**

Exercise 1 (50 points)

A couple of integers (e.g. (1, 2)) is defined by two attributes *a* and *b*.

a) Give the code of the class Couple as follows:
   - Attributes are private,
   - A constructor that initializes these attributes from parameters,
   - A second constructor without parameters, calling the first one, that initializes attributes to default values (0, 0),
   - The method *String toString*() that returns, for example, for a couple(*a*=1, *b*=2) the String (1,2).
   - The mnethod *boolean equals*(*Object o*) inherited from class Object that returns true if attributes are equals one to one, otherwise, it returns false.
   - A static method *cartesianProduct* that, given 2 arrays of integer as parameters, returns an array of couples representing the corresponding cartesian product (e.g. {3,1} × {5,2,4} = {(3,5), (3,2), (3,4), (1,5), (1,2), (1,4)}.

In a *main* method, we have the following instructions:

```
int[] tab1 = {3,1} ;
int[] tab2 = {5,2,4} ;
```

b) Write the instruction that computes the Cartesian product of *tab*1 and *tab*2 and stores it in an array of couples.

Exercise 2 (50 points)

Suppose that we want to implement a simple game that consists on some graphical objects (Circles, Rectangles, Triangles …) moving randomly on a Canvas (Blank screen). These objects have life. The player should click on object many times (e.g. five times) in order to destroy it (make it disappear). The game ends when all objects are destroyed.

To proceed, consider the following classes:

```
public class Circle {
    private double x, y;
    private double radius;
    private int life = 5;

    public Circle(double x, double y, double radius) {
        this.x = x;
        this.y = y;
        this. radius = radius;
    }
```

```
    public void move(double dx, double dy){
        x += dx;
        y += dy;
    }

    public boolean contains(double x, double y){
        //returns true if the distance between the center and the point
        //in parameter is < radius (e.g. point (x,y) is in the circle
        double dx = this.x - x;
        double dy = this.y - y;
        return Math.sqrt(dx*dx + dy*dy) < radius;
    }


    public void getHit(){
        if(life > 0){
            life --;
        }
    }

    public int getLife() {
        return life;
    }
}

public class Rectangle {
    //defined by its upper left corner, width, and length
    private double x, y;
    private double width, length;
    private int life = 5;

    public Rectangle(double x, double y, double width, double length) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.length = length;
    }

    public void move(double dx, double dy){
        x += dx;
        y += dy;
    }

    public boolean contains(double x, double y){
        //returns true if (x, y) is in the rectangle
        double dx = x - this.x;
        double dy = y - this.y;
        return dx >=0 && dy >= 0 && dx <= largeur && dy <= longueur;
    }
```

```
    public void getHit(){
        if(life > 0){
            life --;
        }
    }

    public int getLife () {
        return life;
    }
}

public class Canvas {// classe simplifiée
    private Circle[] setOfCircle = new Circle[100];
    private int nbCircle = 0;
    private Rectangle[] setOfRectangle = new Rectangle[100];
    private int nbRect = 0;

    public void addCircle(Cercle c){
        setOfCircle [nbCircle++] = c;
    }
    //...

    private void removeCircle(Circle c){
        //find c, shift array elements to the left
    }
    //Same thing for Rectangle

    public void attack(double x, double y){
        for(int i =0; i < nbCircle; i++){
            if(setOfCircle[i].contains(x, y)){
                setOfCircle[i].getHit();
            }
            if(setOfCircle[i].getLife() == 0){
                removeCircle(setOfCircle[i]);
            }
        }
        //Same thing for Rectangle
    }
```

You can notice that in class *Canvas*, we should create an array for each type of graphical object, as well as methods to *add* and *remove* these objects. In the method *attack*, we should also implement loops for each type of object. If later we want to use another type of object, we must change the code in class *Canvas* by adding the required instructions to handle the new type.
We want that the class *Canvas* talks to one type of object only and that using new type of objects does not require a change in the class *Canvas*.

a) Make an abstraction of class *Circle* and *Rectangle* by a new class *Form*.
b) Rewrite the class Circle.
c) Rewrite the class *Canvas* using *Form* (It is not required to implement method *remove*)

UNIVERSITÉ LIBANAISE
Faculté des Sciences
Section 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours : I2211 **Année : 2019-2020**
Durée: 1h30' **Exam : Final**

Exercise 1 (40 points) [Class and Object]

Define a class Complex that represents Complex numbers. The class should contain:

- Two private fields *real* and *imag* (double).
- A constructor that initializes these fields by values taken from parameters.
- Another constructor, which uses the first one, that initializes the two fields to 0.
- The *toString* method that returns for a complex object the string *real* + *i imag*.
- A method *module* that returns for a complex object the module of this number (magnitude) ($\sqrt{real^2 + imag^2}$). Use the method *Math.sqrt*(…).
- A method *add* that returns a new Complex object that results from adding the current object to the one passed into parameter. (You should add real parts together and the imaginary parts together.)
- In a *main* method, create two complex numbers (1, 2), (3, 4). Call the method *module* on the second object and indicate which value it would return. Call also the method *add* on the first object by passing the second as a parameter and write down the string that would be returned by the *toString* method of the object that results from the addition.

Exercise 2 (30 points) [Composition]

Consider the following Polynom class:

```
public class Polynom {

    private double[] tab;

    public Polynom(int degree) {
        tab = new double[degree + 1];
    }

    public void setCoeff(int cell, double coeff) {
        tab[cell] = coeff;
    }

    public double compute(double x) {
        double value = 0.0;
        for (int i = 0; i < tab.length; i++) {
            value += tab[i] * Math.pow(x, i);
        }
        return value;
    }

    public static void main(String[] args) {
        //3x^2 - 4x + 1
        Polynom poly = new Polynom(2);
        poly.setCoeff(0, 1);
        poly.setCoeff(1, -4);
        poly.setCoeff(2, 3);

        System.out.println(poly.compute(2));// displays 5
    }
}
```

The class Polynom represents the polynomial $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. The field *tab* would allow to store the different coefficients of the polynomial (e.g. $tab[i] = a_i$). The constructor initializes the length of the array to the value of the polynomial degree + 1.

That said, for the polynomial $3x^{100}-4x+1$, per example, the array should contain 101 slots with just 3 slots that would have non null values.

Consider now the following class Monom :

```
public class Monom {

    private double coeff;
    private int power;

    public Monom(double coeff, int power) {
        this.coeff = coeff;
        this.power = power;
    }

    public double compute(double x) {
        return coeff * Math.pow(x, power);
    }
}
```

This class represents one expression in a polynomial (e.g. in $3x^2-4x+1$, $3x^2$ is a monomial, $-4x$ is another, … The last polynomial contains 3 monomials.) The advantage of using an array of monomials is that the length of the array would correspond exactly to the number of monomials in a polynomial. E.g. for the polynomial $3x^{100}-4x+1$, we need an array of length 3.

a) Define a new class Polynom having an array of Monoms (you should use an index to move within the array). Add to the class:
   - A constructor that initializes the length of the array to the numbers of Monomials.
   - A method *add* that add a Monom passed as parameter to the array of Monomials.
   - A method *compute* which evaluates the polynomial for an x passed as parameter.

b) Write a method *main* in which you create the polynomial $3x^2-4x+1$ using the new class.

Exercise 3 (30 points) [Inheritance, Abstraction and Polymorphism]

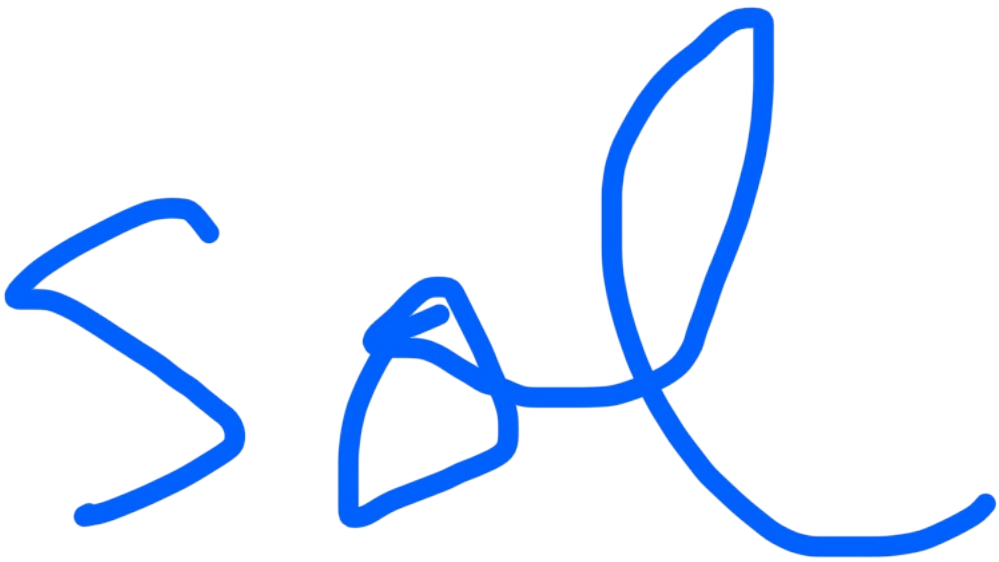
In a University, employees are divided into two categories: staff and instructors. They both have a fixed annual salary. But instructors may have an overload number of hours which are paid following a certain formula that is applied their fixed annual salary. All employees can have their salaries increased by a certain percentage. So we consider the following two classes:

```
public class Staff {

    private String name;
    private double salary;

    public Staff(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public void raiseSalary(int percentage) {
        salary += salary * percentage / 100;
    }


    public double getSalary() {
        return salary;
    }
}
```

```
public class Instructor {

    private String name;
    private double salary;
    private int overload;

    public Instructor(String name, double salary, int overload) {
        this.name = name;
        this.salary = salary;
        this.overload = overload;
    }

    public void raiseSalary(int percentage) {
        salary += salary * percentage / 100;
    }

    public double getSalary() {
        return salary + salary * 12 / 315 * overload;
    }
}
```

The application requires an array in which all employees are stored so that it could compute the sum of all salaries owed to employees.
To do this, you need to build an abstraction (generalization) of these two classes into an Employee class from which the created objects can execute all their methods.

a) Define the class *Employee* and redefine the class Instructor.
b) Define the class University in which you define:
   – An array of *Employee*.
   – A method *add* that adds an employee to the array.
   – A method *totalSalary* that returns the sum of all employees' salaries.
c) Indicate in which line of your code the polymorphism occurs.



Exercise 1

```
public class Complex {

    private double real, imag;

    public Complex(double real, double imag) {
        this.real = real;
        this.imag = imag;
    }

    public Complex() {
        this(0.0, 0.0);
    }

    public String toString() {
        return real + " + i " + imag;
        //or return String.format("%f + i %f", real,imag);
    }

    public double module() {
        return Math.sqrt(real * real + imag * imag);
    }

    public Complex add(Complex c) {
        Complex res = new Complex();
        res.real = this.real + c.real;
        res.imag = this.imag + c.imag;
        return res;
    }

    public static void main(String[] args) {
        Complex c1 = new Complex(1, 2);
        Complex c2 = new Complex(3, 4);
        System.out.println(c2.module()); // 5
        System.out.println(c1.add(c2).toString()); // 4 + i 6
    }
}
```

Exercise 2

```
public class Polynom {

    private Monom[] tab;
    private int index = 0;

    public Polynom(int nombre) {
        tab = new Monom[nombre];
    }

    public void add(Monom monom){
        tab[index++] = monom;
    }

    public double compute(double x) {
        double value = 0.0;
        for (int i = 0; i < tab.length; i++) {
            value += tab[i].compute(x);
        }
        return value;
    }
```

```
    public static void main(String[] args) {
        //3x2-4x+1
        Polynom poly = new Polynom(3);
        poly.add(new Monom(1,0));
        poly.add(new Monom(-4,1));
        poly.add(new Monom(3,2));
    }
}
```

Exercise 3

```
public abstract class Employee {

    protected String name;
    protected double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public void raiseSalary(int percentage) {
        salary += salary * percentage / 100;
    }

    public abstract double getSalary();
}

public class Instructor extends Employee {

    private int overload;

    public Instructor(String name, double salary, int overload) {
        super(name, salary);
        this.overload = overload;
    }

    public double getSalary() {
        return salary + salary * 12 / 315 * overload;
    }
}

public class University {
    private Employee[] tab = new Employee[100];
    private int nb;

    public void add(Employee e){
        tab[nb++] = e;
    }

    public double totalSalary(){
        double total = 0;
        for (int i = 0; i < nb; i++) {
            total += tab[i].getSalary(); // polymorphism
        }
        return total;
    }
}
```

UNIVERSITÉ LIBANAISE
Faculté des Sciences
Section 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours : I2211 — **Année : 2020-2021**
Durée: 1h30' — **Exam : S2**

Exercice 1 (30 points) [Class and Object]

Créer a class *Box* qui est un box :

- 3 attributs longueur, largeur, épaisseur de type *int*;
- Un constructeur qui initialise les attributs à partir des paramètres;
- Un constructeur par défaut qui initialise les attributs par zéro en appelant le premier;
- Une méthode *volume* qui calcule le volume du box;
- Une méthode *peutContenir* qui retourne vrai si et seulement si le box courant peut contenir celui passé en paramètre (c.a.d le volume du box courant est plus grand que celui du box passé en paramètre);
- La méthode *toString* qui retourne une représentation du box: [longueur, largeur, épaisseur];
- La méthode *equals* qui retourne vrai ssi le box courant et celui passé en paramètre ont les attributs égaux.

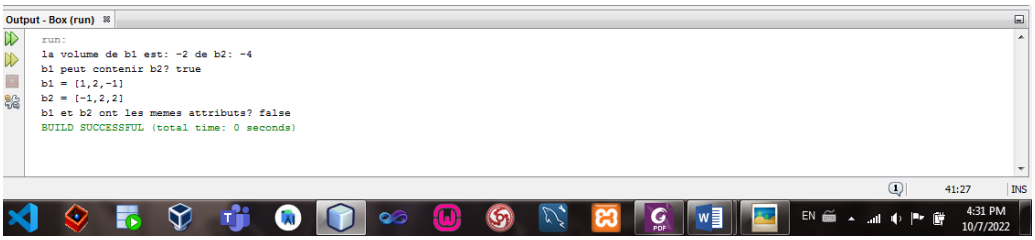
Maintenant, dans une méthode main :

- Créer deux box b1 = [1, 2, –1] et b2 = [–1, 2, 2];
- Calculer et afficher les volumes de b1 et de b2;
- Tester si b1 peut contenir b2.

```
package pkg2emesession;

public class Box {
 private int lon,lar,ep;

   public Box(int lon, int lar, int ep) {
      this.lon = lon;
      this.lar = lar;
      this.ep = ep;
   }

   public Box(){
     this(0,0,0);
   }

   public int volume(){
      return (lon*lar*ep);
```

```
}
public boolean peutContenir(Box b){
   Box c= new Box();
   if(c.volume()>b.volume()) return true;
   else return false;
}

@Override
public String toString() {
   return "[" +lon+"," + lar + "," + ep + ']';
}

public boolean equals(Box b){
  if(b==null) return false;
   if(this==b) return true;
   final Box v=(Box)b;
   return lar==v.lar && lon==v.lon &&ep==v.ep;
}

public static void main(String[] args) {
   Box b1 = new Box(1,2,-1);
   Box b2 = new Box(-1,2,2);
   int v1,v2;
   v1=b1.volume();
   v2=b2.volume();
   b1.peutContenir(b2);
   System.out.println("la volume de b1 est: "+v1+" de b2: "+v2);
   System.out.println("b1 peut contenir b2? "+b1.peutContenir(b2));
   System.out.println("b1 = "+b1.toString());
```

```
        System.out.println("b2 = "+b2.toString());
        System.out.println("b1 et b2 ont les memes attributs?
"+b1.equals(b2));

    }
}
```



Cours : I2211 En **Année : 2020-2021**

Durée: 1h30' **Exam : Final**

Exercise 1 (30 points) [Class and Object]

Define a class *Vector* that represents a vector in the space (three coordinates [x, y, z]):

- Three attributes x, y, z of type *int*;
- A constructor that initializes the attributes from parameters;
- A default constructor that initializes the attributes with zero by calling the previous one;
- A method *norm* that computes the norm (or magnitude) of the vector (square root of $x^2+y^2+z^2$);
- A method *scalarProduct* that returns the scalar product of two vectors: the current vector and the one passed into parameter (scalar product of $[x_1, y_1, z_1]$ and $[x_2, y_2, z_2]$ is equal to $x_1 x_2 + y_1 y_2 + z_1 z_2$);
- A method *vectorProduct* that returns the vector resulting from the vector product of two vectors: the current vector and the one passed into parameter (vector product of $[x_1, y_1, z_1]$ and $[x_2, y_2, z_2]$ is equal to $[y_1 z_2 - z_1 y_2, z_1 x_2 - x_1 z_2, x_1 y_2 - y_1 x_2]$);
- The method *toString* that returns a string representation of a vector: [x, y, z];
- The method *equals* that returns true iff the current vector and the object passed into parameter have equal attributes.

Now, in a main method

- Create the two vectors v1 = [1, 2, –1] and v2 = [–1, 2, 2];
- Compute the vector v3 = vector product of v1 and v2;
- Compute the integer scalar = scalar product of v1 and v2.

Exercise 2 (40 points) [Composition, Exception, List]

Consider the following class Time

```
public class Time {

    private int hour, min;

    public Time(int hour, int min) {
        this.hour = hour;
        this.min = min;
    }

    public String toString() {
        return String.format("%d:%d", hour, min);
    }

    public int toMin() {
        return hour * 60 + min;
    }
}
```

This class represents a Time format without the *second* attribute. The method *toMin* returns the number of minutes since midnight.

a) Rewrite the constructor so it throws an **unchecked** exception of type *InvalidTimeException* (that you have to define) when the parameter *hour* is less than zero or greater than 23 or the parameter *min* is less than zero or greater than 59.

b) Rewrite the header of the class so that objects of this class become comparable (use the requ interface and override its method).

```
public class SubscriptionByCounter {

    private String client;
    private int ampere;
    private int priceAmpere; // minimum price per 1 ampere
    private int kw; // number of kilowatts consumed
    private int priceKw; // price per 1 kw

    public SubscriptionByCounter(String client, int ampere, int priceAmpere,
                                 int kw, int priceKw) {
        this.client = client;
        this.ampere = ampere;
        this.priceAmpere = priceAmpere;
        this.kw = kw;
        this.priceKw = priceKw;
    }

    public String getClient() {
        return client;
    }

    public int computeCost() {
        return ampere * priceAmpere + kw * priceKw;
    }

    public void print() {
        System.out.println(client);
        System.out.println("Ampere: " + ampere + ", Kw: " + kw);
        System.out.println("Cost: " + computeCost());
    }

    public static void main(String[] args) {
        SubscriptionByCounter a = new SubscriptionByCounter("John", 5, 5000,
                                                            250, 1200);
    }
}
```

a) Make an abstraction of these two classes by defining a super class *Subscription* so that we can add any object of these classes to a list of Subscription, manipulate these object as of type Subscription and being able to call methods like *getClient*, *computeCost* and *print*.

b) Rewrite only the class *SubscriptionByCounter* inheriting now from the class *Subscription*.

c) Define a class Company as follows:

```
public class Company {

    //a map associating a client to his subscription
    private Map<String, Subscription> map = new HashMap<>();

    public void add(Subscription sub) {
        // should insert into the map the couple (client, sub) if client
        //does not exist as a key already
    }

    public int totalCost() {
    // sums up costs of all subscriptions and returns it
    }
}
```

Solution examen 2020-2021

Exercice 1:

```
package Ex1;


public class Vecteur {

   private int x,y,z;


    public Vecteur (int x, int y, int z)

{

       this.x=x;

       this.y=y;

       this.z=z;

   }

public Vecteur(){


       this(0,0,0);

   }

public double norm(){


   return Math.sqrt(x*x+y*y+z*z);

}


public int scalarProduct(Vecteur v)

{
```

```
    return x*v.x  +y*v.y +z*v.z;

   }

   public Vecteur vectorProduct(Vecteur v)
{
  int  wx,wy,wz;
      wx=y*v.z -z*v.y;
      wy=z*v.x- x*v.z;
      wz=x*v.y-y*v.x;
      return new Vecteur(wx,wy,wz);
   }

   public String toString(){

      return "["+x+","+y+","+z+"]";
   }
}


public boolean equals(Object o){
      if(o==null) return false;
       if(this==o) return true;
```

```
        final Vecteur v=(Vecteur)o;
        return x==v.x && y==v.y &&z==v.z;
    }

package Ex1;

public class Main {

public static void main(String[] args) {

    Vecteur v1,v2;
    v1=new Vecteur(1,2,-1);
    v2=new Vecteur(-1,2,2);
    Vecteur v3=v1.vectorProduct(v2);
    int sp=v1.scalarProduct(v2);

    System.out.println(v3);

    System.out.println(" scal pro"+sp);
    }

}
```

Exercice 2:

```
public class Time implements Comparable{
public Time(int hour,int min)throws InvalidTimeException
{

      if(hour<0||hour>23||min<0||min>59)
         throw new InvalidTimeException();
      this.hour=hour;
      this.min=min;
   }
public int compareTo(Object o){
  Time t=(Time)   o;
      return toMin()-t.toMin();

   }
```

Exercice 3:

```
public abstract class Subscription {

    private String client;
    private int ampere;
    private int priceAmpere; // minimum price per 1 ampere


    public SubscriptionBy(String client, int ampere, int priceAmpere) {
        this.client = client;
        this.ampere = ampere;
        this.priceAmpere = priceAmpere;

    }

    public String getClient() {
        return client;
    }

    public abstract int computeCost() ;

    public abstract void print() ;

    public static void main(String[] args) {
```

```
        SubscriptionByCounter a = new SubscriptionByCounter("John", 5, 5000,
  250, 1200);
    }
}


public class SubscriptionByCounter extends Subscription {

    private int kw; // number of kilowatts consumed
    private int priceKw; // price per 1 kw


    public SubscriptionByCounter(String client, int ampere, int priceAmpere,  int
kw, int priceKw) {
        super(client,ampere,priceAmpere);
       this.kw = kw;
       this.priceKw = priceKw;
    }
public int computeCost() {
    return ampere * priceAmpere + kw * priceKw;
}


public void print() {
    System.out.println(client);
    System.out.println("Ampere: " + ampere + ", Kw: " + kw);
    System.out.println("Cost: " + computeCost());
```

```
    }


    public static void main(String[] args) {
        SubscriptionByCounter a = new SubscriptionByCounter("John", 5, 5000,
  250, 1200);
    }
}


public class Company {


    //une map associant un client à sa subscription
    private Map<String, Subscription> map = new HashMap<>();


    public void add(Subscription sub) {
        // doit ajouter le couple (client, sub) à la map si le client
  //n'existe pas comme client déjà
  if(!map.containsKey(sub.getClient()))
            map.put(sub.getClient(),sub);
  }


  public int totalCost() {
  // ajoute les couts de toutes les subscriptions et
  //     retourne la valeur
  int couts=0;
```

```
        for(Subscription s : map.values())
                couts+=s.computeCost();

// une autre solution
//ArrayList subscriptions=map.values();
//Iterator its=subscriptions.ietrator();
//for(Subscription s : its)
//                couts+=s.computeCost();
return couts;
        }
}

public class SubscriptionByAmpere extends   Subscription{

    private String client;
    private int ampere; // number of amperes
    private int priceAmpere; // price per 1 ampere

public SubscriptionByAmpere(String client, int ampere, int priceAmpere) {
    super(client,ampere,priceAmpere);
}

public int computeCost() {
    return ampere * priceAmpere;
```

```
}

public void print() {
   System.out.println(client);
   System.out.println("Ampere: " + ampere);
   System.out.println("Cost: " + computeCost());
}

public static void main(String[] args) {
   SubscriptionByAmpere a = new SubscriptionByAmpere("Jack",5,100000);
}}
```

UNIVERSITE LIBANAISE
FACULTE DES SCIENCES
BRANCHE: 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours : I2211 (En) **Date : Oct. 2022**
Session : Final **Time : 1h :30**

In this version, correct answers are all "a", except when the correct answer is "None of the above" then it is "d".

1. Which assertion is the most true about Java ?

   a. Java is a semi-interpreted language.
   b. Java is a compiled language.
   c. Java is an interpreted language.
   d. None of the above.

   Correct answers: a.

   Explanation: Java is compiled into bytecodes then interpreted by JVM.

Sol

2. Which assertion is true about the `int` type in Java ?

   a. It uses 8 bytes for storage.
   b. It uses 2 bytes for storage.
   c. It uses either 2 or 4 bytes for storage depending on the machine.
   d. None of the above.

   Correct answers: d.

   Explanation: `int` type uses 4 bytes for storage.

3. Which assertion is true about the `char` type in Java ?

   a. It uses 2 bytes for storage to handle unicode.
   b. It uses 1 byte for storage as in C-language.
   c. It uses either 1 or 2 bytes for storage depending on the machine.
   d. None of the above.

   Correct answers: a.

   Explanation: ...

4. Which assertion is true about a Java application ?

   a. At least one class should contain a `main` method specified as the main class for the application.
   b. All classes should contain a `main` method.
   c. Only one class should contain a `main` method, otherwise we get runtime error.
   d. None of the above.

   Correct answers: a.

   Explanation: ...

5. What is the output of the following code in `main`?

```
int[] p = {7, 8, 9};
int[] q = new int[2];
q = p; q[2] = 5;
println(p[2]);
```

   a. Compile error.
   b. Runtime error (Array out of bounds)
   c. 9
   d. None of the above.

   Correct answers: d.

   Explanation: `q` refers the same array as `p`. the instruction `q[2] = 5;` is exactly the same as `p[2] = 5;` So the output is 5, no error.

6. What is the output of the following code in `main`?

```
int[] p = {7, 8, 9};
p.length++;
p[3] = 10;
println(p[1]);
```

   a. Compile error.
   b. Runtime error (Array out of bounds)
   c. 8
   d. None of the above.

   Correct answers: a.

   Explanation: `p.length++;` causes a compile error because `length` is public final variable in class array.

[Questions 7–13] Consider the following class.

```
public class Rational {
    private int num, den;

    public Rational(int num, int den) {
    //1
        this.num = num;
        this.den = den;
    }

    public Rational() { //2
        this(0,1);
    }

    public double value(){
        return (double)num/den;
    }
}
```

Consider a `main` method outside the class.

7. In the `main` method, we have the statement `Rational r = new Rational();` Which constructor is called?

   a. the second one then the first.
   b. The first one then the second.
   c. The second one only.
   d. The first one only.

   Correct answers: a.

   Explanation: The second constructor is called first. the instruction `this(0,1);` will call the first constructor.

8. In the `main` method, if we add the statement `println(r.num);` What would be the output?

   a. Compile error.
   b. 0
   c. 1
   d. None of the above.

   Correct answers: a.

   Explanation: `println(r.num);` causes a compile error because `num` is private.

9. In the `main` method, if we add the statement `println(r.value());` What would be the output?

   a. 0
   b. Compile error.
   c. 1
   d. Runtime error (division by 0).

   Correct answers: a.

   Explanation: `r.value();` will return 0/1 = 0.

10. Suppose now, the `main` method is as follow.

```
Rational r = new Rational();
Rational q = new Rational(0, 1);
print(r == q);
println(r.equals(q));
```

    What is the output?

    a. false false.
    b. false true.
    c. true false.
    d. true true.

    Correct answers: a.

    Explanation: `r == q` will return false because different objects.
    `r.equals(q)` will call the method `equals` in class `Object` which, by default, returns `r == q`.

11. We add now, the following method to the class `Rational`.

```
public boolean equals(Rational r) {
   return value() == r.value();
}
```

What is the output of the previous `main`?

a. false true.
b. false false.
c. true false.
d. true true.

Correct answers: a.

Explanation: `r == q` will return false because different objects.
`r.equals(q)` will call the method `equals` in class `Rational` which returns true.

12. We change the `main` method as follow.

```
Object r = new Rational();
Rational q = new Rational(0, 1);
print(q.equals(r));
println(r.equals(q));
```

What is the output now?

a. false false.
b. false true.
c. true false.
d. true true.

Correct answers: a.

Explanation: Now the class `Rational` has two overloaded methods `equals`. One inherited from `Object` taking `Object` as a parameter and one implemented in the class taking `Rational` as a parameter.
`q.equals(r)` calls the inherited method because `r` is `Object` (returns false).
In `r.equals(q)`, the static type of `r` is `Object`, the dynamic type is `Rational`. But `Rational` does not override the method. So `r.equals(q)` calls the method in class `Object`. Polymorphism does not occur in this case.

13. Suppose now, we want to prevent any possible extension to the class `Rational`. What change should we perform to the class?

a. `public final class Rational`
b. `private class Rational`
c. `public static class Rational`
d. `public const class Rational`

Correct answers: a.

Explanation: When a class is final, extension is forbidden.

[Questions 14–16] Consider the following class.

```
public class Dir {

   public final static Dir N = new Dir(0,"North");
   public final static Dir E = new Dir(1,"East");
   public final static Dir S = new Dir(2,"South");
   public final static Dir W = new Dir(3,"West");

   private static Dir tab[] = {N, E, S, W};

   private int index;
   private String name;

   public Dir(int index, String name) {
      this.index = index;
      this.name = name;
   }
```

```
   public String getName() {
      return name;
   }

   public Dir toRight() {
      return tab[(this.index + 1) % tab.length];
   }
}
```

The class `Dir` represents different directions. The four possible objects (N, E, S, W) are created in the class.

14. We want to prevent the user of this class from creating new objects other then those already created in the class. What change should we perform to the class ?

    a. `private Dir(int index, String name)`
    b. `private class Dir`
    c. `public static class Dir`
    d. `public final class Dir`

    Correct answers: a.

    Explanation: The constructor should be private.

15. In a `main` method, we have the following code : `Dir d1 = Dir.W; Dir d2 = d1.toRight(); print(d2.getName());`
    What is the output ?

    a. North
    b. East
    c. South
    d. None of the above

    Correct answers: a.

    Explanation: Index of `d1` is 3. –> (3+1) % 4 = 0. `d1.toRight();` would return `tab[0]`.

16. In a `main` method, we have the following code : `Dir d1 = Dir.N; Dir d2 = d1.toRight(); print(d2);` What is the output ?

    a. East
    b. South
    c. West
    d. None of the above

    Correct answers: d.

    Explanation: Class `Dir` does not override the `toString()` method.

[Questions 17–20] Consider now the following class.

```
public class Robot {
   protected int x, y;
   protected Dir direction;

   public Robot(int x, int y) {
      this.x = x; this.y = y;
      direction = Dir.E;
    }

   public void turnRight() {
      direction = direction.toRight();
   }

   public void move() {
      if (direction == Dir.E) x++;
      else if (direction == Dir.S) y++;
      else if (direction == Dir.W) x--;
      else y--;
    }

   public String toString(){
      return "["+x+","+y+"]: "+direction.
    getName();
   }
}
```

17. In a `main` method, we have the following code :
    `Robot robot = new Robot(1,1); robot.move();`
    `print(robot);` What is the output ?

    a. [2,1] : East
    b. [2,1] : South
    c. [1,2] : East
    d. [1,2] : South

    Correct answers: a.

    Explanation: (x, y, Dir) = (1, 1, E), `robot.move();` increment x.

18. In a `main` method, we have the following code :
    ```
    Robot robot = new Robot(1,1); robot.turnRight();
    robot.turnRight(); robot.move();
    print(robot);
    ```
    What is the output ?

    a. [0,1] : West
    b. [2,1] : West
    c. [1,0] : North
    d. [1,2] : South

    Correct answers: a.

    Explanation: (1, 1, E) –> (1, 1, S) –> (1, 1, W) then (0, 1, W)

19. What is the relationship between the two classes ?

    a. `Robot` *has a* `Dir`
    b. `Dir` *has a* `Robot`
    c. `Robot` *is a* `Dir`
    d. `Dir` *is a* `Robot`

    Correct answers: a.

    Explanation: ...

20. The relationship between the two classes is best described as

    a. Aggregation
    b. Composition
    c. Inheritance
    d. Abstraction

    Correct answers: a.

    Explanation: Aggregation : object referenced by `direction` has its own life cycle (it's not created within the class `Robot`) .

21. Consider the following two assertions : (1) a static method in a class can access instance attributes, (2) an instance method in a class can access class variables (static variables). Are these two assertions

    a. false, true
    b. true, true
    c. false, false
    d. true, false

    Correct answers: a.

    Explanation: ...

[Questions 22–24] Consider now the following class which represents a Strong Robot that extends Robot but in addition, it can move faster (increment and decrement x and y by 2) and can turn left.

```
public class StrongRobot extends Robot{

   public StrongRobot(int x, int y) {
      this.x = x; this.y = y;
      direction = Dir.E;
   }

   public void move(){
      if (direction == Dir.E) x += 2;
      else if (direction == Dir.S) y += 2;
      else if (direction == Dir.W) x -= 2;
      else y -= 2;
    }

   public void turnLeft(){
      turnRight();turnRight();turnRight();
   }
}
```

22. The class `StrongRobot` shows a compile error in the constructor. Why ?

    a. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `super(x, y);`
    b. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `new Robot(x, y);`
    c. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `this(x, y);`
    d. Because the attributes (x, y, direction) are protected in `Robot` so `StrongRobot` can't access them.

    Correct answers: a.

    Explanation: ...

23. We want to change the implementation of `move` in class `StrongRobot` by calling the one implemented in `Robot`. This can be achieved by the following instructions

    a. `super.move(); super.move();`
    b. `((Robot)this).move(); ((Robot)this).move();`
    c. `move(); move();`
    d. None of the above

    Correct answers: a.

    Explanation: ...

24. We want to change the implementation of the method `turnLeft` so it accesses directly the inherited attribute `direction`. This can be achieved by the following instructions

    a. `direction = direction.toRight().toRight().toRight();`
    b. `direction = direction.toRight() = direction. toRight() = direction.toRight();`
    c. `direction.toRight(); direction.toRight(); direction.toRight();`
    d. None of the above

    Correct answers: a.

    Explanation: `direction.toRight()` returns an object of type `Dir`
    In part (b) there is a compile error (assigning value to value !)
    In part (c) `direction.toRight();` does not change the value of `direction`.

[Questions 25–29] Suppose now that the class `StrongRobot` works correctly and Consider the following `main` method.

```
StrongRobot sro = new StrongRobot(2,2);
Robot rosro = new StrongRobot(3,3);
```

25. Suppose we add the instructions `sro.turnLeft(); sro.move(); print(sro);` What is the output ?

    a. [2,0] : North
    b. [2,1] : North
    c. [0,2] : North
    d. Compile error.

    Correct answers: a.

    Explanation: ...

26. Suppose we add the instructions `rosro.turnLeft(); rosro.move(); print(rosro);` What is the output ?

    a. Compile error.
    b. [3,1] : North
    c. [1,3] : West
    d. [3,1] : West

    Correct answers: a.

    Explanation: When a method is called, the compiler verifies the existence of the method in the *static* type of the object which, in this case, is `Robot`. (`Robot` does not have `turnLeft()` method)

27. Suppose we add the instructions `rosro.turnRight();`
    `rosro.move(); print(rosro);` What is the output ?

    a. [3,5] : South
    b. [3,4] : South
    c. [2,4] : South
    d. Compile error.

    Correct answers: a.

    Explanation: ...

28. Which one of the following statements is true ?

    a. Method `move()` in `StrongRobot` is overriding the one in `Robot`
    b. Method `move()` in `StrongRobot` is overloading the one in `Robot`
    c. Method `turnRight()` in `StrongRobot` is overloading the one in `Robot`
    d. Method `turnLeft()` in `StrongRobot` is overriding method `turnRight()` in `Robot`

    Correct answers: a.

    Explanation: ...

29. When compiling the instruction
    `Robot rosro = new StrongRobot(3,3);` the compiler will implicitly perform

    a. an upcasting from `StrongRobot` to `Robot`.
    b. a downcasting from `Robot` to `StrongRobot`.
    c. an upcasting from `Robot` to `StrongRobot`.
    d. a downcasting from `StrongRobot` to `Robot`.

    Correct answers: a.

Explanation: ...

30. Which one of the following assertions is true about *polymorphism* (runtime polymorphism) ?

    a. Polymorphism is about binding method call to the corresponding overridden method depending on the dynamic type of the current object.
    b. Polymorphism is about binding method call to the corresponding overloaded method depending on the dynamic type of the current object.
    c. Polymorphism is about binding method call to the corresponding overridden method depending on the static type of the current object.
    d. Polymorphism is about binding method call to the corresponding overloaded method depending on the static type of the current object.

    Correct answers: a.

    Explanation: ...

31. Consider the following class

```
public abstract class A {
   private int num;

   public A(int num) {
      this.num = num;
   }

   public void mm(int p){
      num += p;
   }
}
```

    Is there a compile error ?

    a. No, the class compiles without error.
    b. Yes, an abstract class should have an abstract method.
    c. Yes, an abstract class should not have a constructor.
    d. Yes, an abstract class should not have a private attribute.

    Correct answers: a.

    Explanation: ...

32. Consider the following classes

```
public abstract class NewA {
   protected int num;

   public abstract void mm(int p);
}
```

```
public abstract class B extends NewA {

   public void tt(int p) {
      num += p;
   }
}
```

Is there a compile error?

a. No, the classes compile without error.
b. Yes, an abstract class cannot extend another abstract class.
c. Yes, class `B` should implement method `mm`.
d. Yes, method `tt` cannot access the inherited attribute `num`.

Correct answers: a.

Explanation: When an abstract class B extends another abstract class A, it is not required for B to implement the abstract methods of A.

33. Consider now the following class `C`

```
public class C extends NewA {
    public C(int n) {
      num = n;
   }
   public void mm(int p) {
      num += p;
   }
}
```

In a `main` method we have the instruction `NewA a = new C(5); print(a);` What is the output?

a. Compile error, because `a` is of type abstract class.
b. Compile error, because there is no call to the super constructor.
c. 5
d. None of the above.

Correct answers: d.

Explanation: Nothing wrong with the code. The class does not not override `toString()`.

34. In a `main` method we have the instruction `NewA a = new NewA();` Is there a compile error?

a. Yes, because no object can be created from an abstract class.
b. Yes, because there is no default constructor in the class.
c. No, it compiles fine because default constructor is provided implicitly.
d. None of the above

Correct answers: a.

Explanation: ...

35. Java *Grabage Collector*

    a. runs asynchronously on the heap to free memory used by objects that are no longer referenced.
    b. is always running on the heap to free memory used by objects that are no longer referenced.
    c. runs asynchronously on the stack to free memory used by objects that are no longer referenced.
    d. runs asynchronously on the heap to free memory used by those objects whose classes override the method `finalize`, that are no longer referenced.

    Correct answers: a.

UNIVERSITE LIBANAISE
FACULTE DES SCIENCES
BRANCHE: 3

الجامعة اللبنانية
كلية العلوم
الفرع الثالث

Cours : I2211 (En) **Date : Oct. 2022**
Session : Final **Time : 1h :30**

In this version, correct answers are all "a", except when the correct answer is "None of the above" then it is "d".

1. Which assertion is the most true about Java ?

   a. Java is a semi-interpreted language.
   b. Java is a compiled language.
   c. Java is an interpreted language.
   d. None of the above.

   Correct answers: a.

   Explanation: Java is compiled into bytecodes then interpreted by JVM.

Sol

2. Which assertion is true about the `int` type in Java ?

   a. It uses 8 bytes for storage.
   b. It uses 2 bytes for storage.
   c. It uses either 2 or 4 bytes for storage depending on the machine.
   d. None of the above.

   Correct answers: d.

   Explanation: `int` type uses 4 bytes for storage.

3. Which assertion is true about the `char` type in Java ?

   a. It uses 2 bytes for storage to handle unicode.
   b. It uses 1 byte for storage as in C-language.
   c. It uses either 1 or 2 bytes for storage depending on the machine.
   d. None of the above.

   Correct answers: a.

   Explanation: ...

4. Which assertion is true about a Java application ?

   a. At least one class should contain a `main` method specified as the main class for the application.
   b. All classes should contain a `main` method.
   c. Only one class should contain a `main` method, otherwise we get runtime error.
   d. None of the above.

   Correct answers: a.

   Explanation: ...

5. What is the output of the following code in `main`?

```
int[] p = {7, 8, 9};
int[] q = new int[2];
q = p; q[2] = 5;
println(p[2]);
```

   a. Compile error.
   b. Runtime error (Array out of bounds)
   c. 9
   d. None of the above.

   Correct answers: d.

   Explanation: `q` refers the same array as `p`. the instruction `q[2] = 5;` is exactly the same as `p[2] = 5;` So the output is 5, no error.

6. What is the output of the following code in `main`?

```
int[] p = {7, 8, 9};
p.length++;
p[3] = 10;
println(p[1]);
```

   a. Compile error.
   b. Runtime error (Array out of bounds)
   c. 8
   d. None of the above.

   Correct answers: a.

   Explanation: `p.length++;` causes a compile error because `length` is public final variable in class array.

[Questions 7–13] Consider the following class.

```
public class Rational {
    private int num, den;

    public Rational(int num, int den) {
    //1
        this.num = num;
        this.den = den;
    }

    public Rational() { //2
        this(0,1);
    }

    public double value(){
        return (double)num/den;
    }
}
```

Consider a `main` method outside the class.

7. In the `main` method, we have the statement `Rational r = new Rational();` Which constructor is called?

   a. the second one then the first.
   b. The first one then the second.
   c. The second one only.
   d. The first one only.

   Correct answers: a.

   Explanation: The second constructor is called first. the instruction `this(0,1);` will call the first constructor.

8. In the `main` method, if we add the statement `println(r.num);` What would be the output?

   a. Compile error.
   b. 0
   c. 1
   d. None of the above.

   Correct answers: a.

   Explanation: `println(r.num);` causes a compile error because `num` is private.

9. In the `main` method, if we add the statement `println(r.value());` What would be the output?

   a. 0
   b. Compile error.
   c. 1
   d. Runtime error (division by 0).

   Correct answers: a.

   Explanation: `r.value();` will return 0/1 = 0.

10. Suppose now, the `main` method is as follow.

```
Rational r = new Rational();
Rational q = new Rational(0, 1);
print(r == q);
println(r.equals(q));
```

    What is the output?

    a. false false.
    b. false true.
    c. true false.
    d. true true.

    Correct answers: a.

    Explanation: `r == q` will return false because different objects.
    `r.equals(q)` will call the method `equals` in class `Object` which, by default, returns `r == q`.

11. We add now, the following method to the class `Rational`.

```
public boolean equals(Rational r) {
   return value() == r.value();
}
```

What is the output of the previous `main`?

a. false true.
b. false false.
c. true false.
d. true true.

Correct answers: a.

Explanation: `r == q` will return false because different objects.
`r.equals(q)` will call the method `equals` in class `Rational` which returns true.

12. We change the `main` method as follow.

```
Object r = new Rational();
Rational q = new Rational(0, 1);
print(q.equals(r));
println(r.equals(q));
```

What is the output now?

a. false false.
b. false true.
c. true false.
d. true true.

Correct answers: a.

Explanation: Now the class `Rational` has two overloaded methods `equals`. One inherited from `Object` taking `Object` as a parameter and one implemented in the class taking `Rational` as a parameter.
`q.equals(r)` calls the inherited method because `r` is `Object` (returns false).
In `r.equals(q)`, the static type of `r` is `Object`, the dynamic type is `Rational`. But `Rational` does not override the method. So `r.equals(q)` calls the method in class `Object`. Polymorphism does not occur in this case.

13. Suppose now, we want to prevent any possible extension to the class `Rational`. What change should we perform to the class?

a. `public final class Rational`
b. `private class Rational`
c. `public static class Rational`
d. `public const class Rational`

Correct answers: a.

Explanation: When a class is final, extension is forbidden.

[Questions 14–16] Consider the following class.

```
public class Dir {

   public final static Dir N = new Dir(0,"North");
   public final static Dir E = new Dir(1,"East");
   public final static Dir S = new Dir(2,"South");
   public final static Dir W = new Dir(3,"West");

   private static Dir tab[] = {N, E, S, W};

   private int index;
   private String name;

   public Dir(int index, String name) {
      this.index = index;
      this.name = name;
   }
```

```
   public String getName() {
      return name;
   }

   public Dir toRight() {
      return tab[(this.index + 1) % tab.length];
   }
}
```

The class `Dir` represents different directions. The four possible objects (N, E, S, W) are created in the class.

14. We want to prevent the user of this class from creating new objects other then those already created in the class. What change should we perform to the class ?

    a. `private Dir(int index, String name)`
    b. `private class Dir`
    c. `public static class Dir`
    d. `public final class Dir`

    Correct answers: a.

    Explanation: The constructor should be private.

15. In a `main` method, we have the following code : `Dir d1 = Dir.W; Dir d2 = d1.toRight(); print(d2.getName());`
    What is the output ?

    a. North
    b. East
    c. South
    d. None of the above

    Correct answers: a.

    Explanation: Index of `d1` is 3. –> (3+1) % 4 = 0. `d1.toRight();` would return `tab[0]`.

16. In a `main` method, we have the following code : `Dir d1 = Dir.N; Dir d2 = d1.toRight(); print(d2);` What is the output ?

    a. East
    b. South
    c. West
    d. None of the above

    Correct answers: d.

    Explanation: Class `Dir` does not override the `toString()` method.

[Questions 17–20] Consider now the following class.

```
public class Robot {
   protected int x, y;
   protected Dir direction;

   public Robot(int x, int y) {
      this.x = x; this.y = y;
      direction = Dir.E;
    }

   public void turnRight() {
      direction = direction.toRight();
   }

   public void move() {
      if (direction == Dir.E) x++;
      else if (direction == Dir.S) y++;
      else if (direction == Dir.W) x--;
      else y--;
    }

   public String toString(){
      return "["+x+","+y+"]: "+direction.
    getName();
   }
}
```

17. In a `main` method, we have the following code :
    `Robot robot = new Robot(1,1); robot.move();`
    `print(robot);` What is the output ?

    a. [2,1] : East
    b. [2,1] : South
    c. [1,2] : East
    d. [1,2] : South

    Correct answers: a.

    Explanation: (x, y, Dir) = (1, 1, E), `robot.move();` increment x.

18. In a `main` method, we have the following code :
    ```
    Robot robot = new Robot(1,1); robot.turnRight();
    robot.turnRight(); robot.move();
    print(robot);
    ```
    What is the output ?

    a. [0,1] : West
    b. [2,1] : West
    c. [1,0] : North
    d. [1,2] : South

    Correct answers: a.

    Explanation: (1, 1, E) –> (1, 1, S) –> (1, 1, W) then (0, 1, W)

19. What is the relationship between the two classes ?

    a. `Robot` *has a* `Dir`
    b. `Dir` *has a* `Robot`
    c. `Robot` *is a* `Dir`
    d. `Dir` *is a* `Robot`

    Correct answers: a.

    Explanation: ...

20. The relationship between the two classes is best described as

    a. Aggregation
    b. Composition
    c. Inheritance
    d. Abstraction

    Correct answers: a.

    Explanation: Aggregation : object referenced by `direction` has its own life cycle (it's not created within the class `Robot`) .

21. Consider the following two assertions : (1) a static method in a class can access instance attributes, (2) an instance method in a class can access class variables (static variables). Are these two assertions

    a. false, true
    b. true, true
    c. false, false
    d. true, false

    Correct answers: a.

    Explanation: ...

[Questions 22–24] Consider now the following class which represents a Strong Robot that extends Robot but in addition, it can move faster (increment and decrement x and y by 2) and can turn left.

```
public class StrongRobot extends Robot{

   public StrongRobot(int x, int y) {
      this.x = x; this.y = y;
      direction = Dir.E;
   }

   public void move(){
      if (direction == Dir.E) x += 2;
      else if (direction == Dir.S) y += 2;
      else if (direction == Dir.W) x -= 2;
      else y -= 2;
    }

   public void turnLeft(){
      turnRight();turnRight();turnRight();
   }
}
```

22. The class `StrongRobot` shows a compile error in the constructor. Why ?

    a. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `super(x, y);`
    b. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `new Robot(x, y);`
    c. Because there should be a call to the constructor of the class `Robot` which can be achieved by the instruction `this(x, y);`
    d. Because the attributes (x, y, direction) are protected in `Robot` so `StrongRobot` can't access them.

    Correct answers: a.

    Explanation: ...

23. We want to change the implementation of `move` in class `StrongRobot` by calling the one implemented in `Robot`. This can be achieved by the following instructions

    a. `super.move(); super.move();`
    b. `((Robot)this).move(); ((Robot)this).move();`
    c. `move(); move();`
    d. None of the above

    Correct answers: a.

    Explanation: ...

24. We want to change the implementation of the method `turnLeft` so it accesses directly the inherited attribute `direction`. This can be achieved by the following instructions

    a. `direction = direction.toRight().toRight().toRight();`
    b. `direction = direction.toRight() = direction. toRight() = direction.toRight();`
    c. `direction.toRight(); direction.toRight(); direction.toRight();`
    d. None of the above

    Correct answers: a.

    Explanation: `direction.toRight()` returns an object of type `Dir`
    In part (b) there is a compile error (assigning value to value !)
    In part (c) `direction.toRight();` does not change the value of `direction`.

[Questions 25–29] Suppose now that the class `StrongRobot` works correctly and Consider the following `main` method.

```
StrongRobot sro = new StrongRobot(2,2);
Robot rosro = new StrongRobot(3,3);
```

25. Suppose we add the instructions `sro.turnLeft(); sro.move(); print(sro);` What is the output ?

    a. [2,0] : North
    b. [2,1] : North
    c. [0,2] : North
    d. Compile error.

    Correct answers: a.

    Explanation: ...

26. Suppose we add the instructions `rosro.turnLeft(); rosro.move(); print(rosro);` What is the output ?

    a. Compile error.
    b. [3,1] : North
    c. [1,3] : West
    d. [3,1] : West

    Correct answers: a.

    Explanation: When a method is called, the compiler verifies the existence of the method in the *static* type of the object which, in this case, is `Robot`. (`Robot` does not have `turnLeft()` method)

27. Suppose we add the instructions `rosro.turnRight();`
    `rosro.move(); print(rosro);` What is the output ?

    a. [3,5] : South
    b. [3,4] : South
    c. [2,4] : South
    d. Compile error.

    Correct answers: a.

    Explanation: ...

28. Which one of the following statements is true ?

    a. Method `move()` in `StrongRobot` is overriding the one in `Robot`
    b. Method `move()` in `StrongRobot` is overloading the one in `Robot`
    c. Method `turnRight()` in `StrongRobot` is overloading the one in `Robot`
    d. Method `turnLeft()` in `StrongRobot` is overriding method `turnRight()` in `Robot`

    Correct answers: a.

    Explanation: ...

29. When compiling the instruction
    `Robot rosro = new StrongRobot(3,3);` the compiler will implicitly perform

    a. an upcasting from `StrongRobot` to `Robot`.
    b. a downcasting from `Robot` to `StrongRobot`.
    c. an upcasting from `Robot` to `StrongRobot`.
    d. a downcasting from `StrongRobot` to `Robot`.

    Correct answers: a.

Explanation: ...

30. Which one of the following assertions is true about *polymorphism* (runtime polymorphism) ?

    a. Polymorphism is about binding method call to the corresponding overridden method depending on the dynamic type of the current object.
    b. Polymorphism is about binding method call to the corresponding overloaded method depending on the dynamic type of the current object.
    c. Polymorphism is about binding method call to the corresponding overridden method depending on the static type of the current object.
    d. Polymorphism is about binding method call to the corresponding overloaded method depending on the static type of the current object.

    Correct answers: a.

    Explanation: ...

31. Consider the following class

```
public abstract class A {
   private int num;

   public A(int num) {
      this.num = num;
   }

   public void mm(int p){
      num += p;
   }
}
```

    Is there a compile error ?

    a. No, the class compiles without error.
    b. Yes, an abstract class should have an abstract method.
    c. Yes, an abstract class should not have a constructor.
    d. Yes, an abstract class should not have a private attribute.

    Correct answers: a.

    Explanation: ...

32. Consider the following classes

```
public abstract class NewA {
   protected int num;

   public abstract void mm(int p);
}
```

```
public abstract class B extends NewA {

   public void tt(int p) {
      num += p;
   }
}
```

Is there a compile error?

a. No, the classes compile without error.
b. Yes, an abstract class cannot extend another abstract class.
c. Yes, class `B` should implement method `mm`.
d. Yes, method `tt` cannot access the inherited attribute `num`.

Correct answers: a.

Explanation: When an abstract class B extends another abstract class A, it is not required for B to implement the abstract methods of A.

33. Consider now the following class `C`

```
public class C extends NewA {
    public C(int n) {
      num = n;
   }
   public void mm(int p) {
      num += p;
   }
}
```

In a `main` method we have the instruction `NewA a = new C(5); print(a);` What is the output?

a. Compile error, because `a` is of type abstract class.
b. Compile error, because there is no call to the super constructor.
c. 5
d. None of the above.

Correct answers: d.

Explanation: Nothing wrong with the code. The class does not not override `toString()`.

34. In a `main` method we have the instruction `NewA a = new NewA();` Is there a compile error?

a. Yes, because no object can be created from an abstract class.
b. Yes, because there is no default constructor in the class.
c. No, it compiles fine because default constructor is provided implicitly.
d. None of the above

Correct answers: a.

Explanation: ...

35. Java *Grabage Collector*

    a. runs asynchronously on the heap to free memory used by objects that are no longer referenced.
    b. is always running on the heap to free memory used by objects that are no longer referenced.
    c. runs asynchronously on the stack to free memory used by objects that are no longer referenced.
    d. runs asynchronously on the heap to free memory used by those objects whose classes override the method `finalize`, that are no longer referenced.

    Correct answers: a.