Verificare 01

Petculescu Mihai-Silviu

Exerciţiu 1

Creaţi o clasă cu un constructor privat. Vedeţi ce se întâmplă la compilare dacă creaţi o instanţă a clasei într-o metodă `main`.

```
public class E1 {
  public static void main(String[] args) {
    Rectangle r1 = new Rectangle();
    System.out.println(r1);
  }
}
class Rectangle {
  int length, width;
  private Rectangle() {
    this.length = this.width = 0;
  }
  @Override
  public String toString() {
    return String.format("Rectangle: %d length, %d width.", length, width);
  }
}
```

Va rezulta o eroare de compilare, deoarece constructorul apelat este privat. (`Rectangle() has private access in Rectangle`)

Exerciţiu 2

Creaţi o clasă ce conţine două atribute nestatice private, un `int` şi un `char` care nu sunt iniţializate şi tipăriţi valorile acestora pentru a verifica dacă Java realizează iniţializarea implicită.

```
public class E2 {
  public static void main(String[] args) {
    Object1 obj1 = new Object1();
    System.out.print(obj1);
  }
}

class Object1 {
```

```
  private int n;
  private char c;
  @Override
  public String toString() {
    return String.format("'%c', %d", c, n);
  }
}
```

Se va afişa: `'', 0`, deci atributele private `c` şi `n` sunt iniţializate implicit de Java.

Exerciţiu 3

```
class Motor {
  private int capacitate;
  public Motor(int c) {
    capacitate = c;
  }
  public void setCapacitate(int c) {
    capacitate = c;
  }
  public void tipareste() {
    System.out.println("Motor de capacitate " + capacitate);
  }
}
```

Fiind dată implementarea clasei `Motor`, se cere să se precizeze ce se va afişa în urma rulări secvenţei:

```
Motor m1, m2;
m1 = new Motor(5);
m2 = m1;
m2.setCapacitate(10);
m1.tipareste();
```

Se va afişa: `Motor de capacitate 10`. Alocam în memorie 2 referinţe către obiecte de tip `Motor`, `m1` şi `m2`, la care asociem o instanţă a clasei (la amândouă aceeaşi instanţă). Astfel apelând `m2.setCapacitate(2)` are ca efect modificarea atributului `capacitate` a instanţei create şi cum amandouă obiectele `m1` şi `m2` pointeaza la aceasta, se va afişa valoarea 10.

Exerciţiu 4

Un sertar este caracterizat de o lăţime, lungime şi înalţime. Un birou are două sertare şi, evident, o lăţime, lungime şi înalţime. Creaţi clasele `Sertar` şi `Birou` corespunzătoare specificaţiilor de mai sus. Creaţi pentru fiecare clasă constructorul potrivit astfel încât carateristicile instanţelor să fie setate la crearea acestora. Clasa `Sertar` conţine o metodă `tipareste` al cărei apel va produce tipărirea pe ecran a sertarului sub forma $'Sertar' + l + L + H$, unde l, L, H sunt valorile coresupunzătoare lăţimii, lungimii şi înalţimii sertarului. Clasa `Birou` conţine o metodă `tipareste` cu ajutorul căreia se vor tipări toate componentele biroului. Creaţi într-o metodă `main` două sertare, un birou şi tipăriţi componentele biroului.

```
public class E3 {
  public static void main(String[] args) {
    Sertar s1 = new Sertar(4, 5, 2);
    Sertar s2 = new Sertar(5, 10, 2);
```

```
    Birou b = new Birou(25, 50, 30, s1, s2);
    b.tipareste();
  }
}

class Sertar {
  int l, L, H;
  Sertar(int l, int L, int H) {
    this.l = l;
    this.L = L;
    this.H = H;
  }
  public void tipareste() {
    String s_print = String.format("'Sertar' + %d + %d + %d", l, L, H);
    System.out.println(s_print);
  }
}

class Birou {
  Sertar s1, s2;
  int l, L, H;
  Birou(int l, int L, int H, Sertar s1, Sertar s2) {
    this.l = l;
    this.L = L;
    this.H = H;
    this.s1 = s1;
    this.s2 = s2;
  }
  public void tipareste() {
    String s_print = String.format("'Birou' + %d + %d + %d", l, L, H);
    System.out.println(s_print);
    s1.tipareste();
    s2.tipareste();
  }
}
```

```
'Birou' + 25 + 50 + 30
'Sertar' + 4 + 5 + 2
'Sertar' + 5 + 10 + 2
```

Exerciţiu 5

Definiţi o clasă `Complex` care modelează lucrul cu numere complexe. Membrii acestei clase sunt:

- două atribute de tip `double` pentru părţile reală, respectiv imaginară ale numărului complex
- un constructor cu doi parametri de tip `double`, pentru setarea celor două părţi ale numărului (reală şi imaginară)
- o metodă de calcul a modulului numărului complex. Se precizează că modulul unui număr complex este egal cu radical din ($re \cdot re + img \cdot img$) unde re este partea reală, iar img este partea imaginară. Pentru calculul radicalului se va folosi metoda statică predefinită `Math.sqrt` care necesită un parametru de tip double şi returneaza tot un `double`
- o metodă de afişare pe ecran a valorii numărului complex, sub forma $re + i \cdot im$
- o metodă care returnează suma dintre două obiecte complexe. Această metodă are un parametru de tip `Complex` şi returnează suma dintre obiectul curent (obiectul care oferă

serviciul de adunare) şi cel primit ca parametru. Tipul returnat de această metodă este `Complex`.
- o metodă care returnează de câte ori s-au afişat pe ecran numere complexe.

Pe lângă clasa `Complex` se va defini o clasă `ClientComplex` care va conţine într-o metoda `main` exemple de utilizare ale metodelor clasei `Complex`.

```
public class ClientComplex {
  public static void main(String[] args) {
    Complex c1 = new Complex(2, 4);
    Complex c2 = new Complex(3, 5);

    System.out.print("c1: ");
    c1.Tiparire();
    System.out.print("c2: ");
    c2.Tiparire();

    System.out.println("Modul c1: " + c1.Modul());
    System.out.println("Suma c1 + c2: ");
    c1.Suma(c2).Tiparire();

    Complex.StatisticiAfisare();
  }
}

class Complex {
  double re, img;
  static int k_print = 0;

  Complex(double re, double img) {
    this.re = re;
    this.img = img;
  }
  double Modul() {
    return Math.sqrt(re * re + img * img);
  }
  void Tiparire() {
    k_print++;
    System.out.println(re + " + i * " + img);
  }
  Complex Suma(Complex x) {
    return new Complex(x.re + re, x.img + img);
  }
  static void StatisticiAfisare() {
    System.out.println("S-au afisat de " + k_print + " ori.");
  }
}
```

```
c1: 2.0 + i * 4.0
c2: 3.0 + i * 5.0
Modul c1: 4.47213595499958
Suma c1 + c2:
5.0 + i * 9.0
S-au afisat de 3 ori.
```