

3. Implicit, metoda equals din clasa Object realizează: *
(5 Points)

- ☒ comparația a două obiecte din punct de vedere al referințelor lor
- ☐ nu exista metoda equals în clasa Object
- ☐ comparația a două obiecte din punct de vedere structural
- ☐ o clonă a obiectului curent

☐ o clonă a obiectului curent

4. Modificatorul de acces al unei metode redefinite într-o subclasă: *
(5 Points)

- ☐ trebuie să fie același cu cel din superclasă
- ☐ trebuie să fie mereu protected
- ☒ nu trebuie să fie mai restrictiv decât cel al metodei din superclasă
- ☐ poate fi orice modifier

5. Definim următoarea metodă care trebuie să efectueze o anumită prelucrare asupra elementelor de tip Persoana dintr-un tablou care îndeplinesc un anumit criteriu:

```
void modificare(Persoana[] persoane, Tip_1 criteriu, Tip_2 prelucrare) {  
    for(Persoana p:persoane)  
        if(criteriu.test(p))  
            prelucrare.accept(p);  
}
```

Descriptorii corespunzători celor 2 parametrii criteriu și prelucrare trebuie să fie: *
(5 Points)

- ☐ Tip_1 = Function și Tip_2 = Supplier
- ☐ Tip_1 = Supplier și Tip_2 = Predicate
- ☐ Tip_1 = Predicate și Tip_2 = Function
- ☒ Tip_1 = Predicate și Tip_2 = Consumer

6. Dacă în blocul try - catch se folosește instrucțiunea return atunci: *
(5 Points)

- ☐ blocul finally nu se mai execută
- ☒ blocul finally se execută
- ☐ în blocul try - catch nu se poate folosi instrucțiunea return
- ☐ blocul try - catch nu poate să conțină blocul finally în acest caz

7. Pentru a realiza o relație compozitie de tip HAS A, atunci: *

7. Pentru a realiza o relație compoziție de tip HAS_A, atunci: *
(5 Points)

- ☐ se va încapsula în clasa container o referințe spre obiectul asociat
- ☒ se va încapsula în clasa container o referință a unei copii locale a obiectului asociat
- ☐ se va extinde clasa container
- ☐ se va implementa o interfață de către clasa container

8. Crearea unei clase imutabile trebuie să respecte următoarele reguli: *

8. Crearea unei clase imutabile trebuie să respecte următoarele reguli: *
(5 Points)

- ☒ toate câmpurile clasei vor fi declarate ca fiind final și private
- ☒ clasa nu va conține metode de tip set sau alte metode care pot modifica valorile câmpurilor
- ☐ clasa va permite rescrierea metodelor sale
- ☒ nu se vor folosi referințe spre obiecte externe, ci spre copii ale lor

9. Considerând faptul că șirul de caractere s conține mai multe cuvinte despărțite între ele prin c

9. Considerând faptul că șirul de caractere *s* conține mai multe cuvinte despărțite între ele prin cel puțin două spații, cum va modifica următoarea secvență de cod șirul de caractere *s*?

```
int p = s.indexOf(" ");  
while(p != -1)  
{  
    s = s.substring(0, p) + s.substring(p+1);  
    p = s.indexOf(" ");  
} *
```

(5 Points)

- ☒ vor fi eliminate toate spațiile
- ☐ din fiecare cuvânt va fi eliminată prima literă a sa
- ☐ din fiecare grup de cel puțin două spații dintre două cuvinte va fi eliminat doar primul spațiu
- ☐ nu va modifica deloc

☐ nu va modifica deloc


10. Dacă o clasă serializabilă extinde o clasă neserializabilă, atunci: *
(5 Points)

- ☐ subclasa este serializabilă în mod implicit
- ☐ subclasa trebuie să implementeze interfața Serializable, în caz contrar se obține o excepție la compilare
- ☐ subclasa trebuie să implementeze interfața Serializable, în caz contrar se obține o excepție la executare
- ☒ datele membre accesibile ale superclasei nu vor fi serializate

11. Pentru o colecție de tip HashMap următoarele afirmații sunt adevărate: *
(5 Points)


11. Pentru o colecție de tip HashMap următoarele afirmații sunt adevărate: *
(5 Points)

- ☐ se menține ordinea de inserare și se poate stabili o anumită ordine a perechilor
- ☒ este permisă utilizarea valorii null atât pentru cheie, cât și pentru valoare.
- ☒ se poate asocia aceeași valoare mai multor chei.
- ☐ se permite duplicarea cheilor

12. Care dintre următoarele afirmații despre metoda forEach sunt adevărate? * 

☒ se poate asocia aceeași valoare mai multor chei.

☐ se permite duplicarea cheilor

12. Care dintre următoarele afirmații despre metoda `forEach` sunt adevărate? * 
(5 Points)


☒ are un parametru de tip `Consumer`

☒ parcurge secvențial o colecție

☐ permite modificarea elementului curent dintr-o colecție

☒ nu returnează nicio valoare

13. Considerând `lcuv` o listă de siruri de caractere, care dintre următoarele expresii utilizează corect o

13. Considerând `lcuv` o listă de șiruri de caractere, care dintre următoarele expresii utilizează corect o referință spre o metodă? ★ 
(5 Points)

- ☐ `lcuv.replaceAll(s::toUpperCase());`
- ☒ `lcuv.replaceAll(String::toUpperCase);`
- ☐ `lcuv.replaceAll(String::toUpperCase());`
- ☐ `lcuv.replaceAll(s::toUpperCase());`

14. Considerăm următorul program Java:

```
public class Test {  
    public static void main(String args[]) throws FileNotFoundException {  
        Scanner fin = new Scanner(new File("exemplu.txt"));  
        int s = 0;  
        try{  
            while(fin.hasNext())  
            {  
                int x = fin.nextInt();  
                s = s + x;  
            }  
        }catch(ArithmeticException ex) {  
            System.out.println("Număr incorect!");  
        }  
        finally {  
            System.out.println(s);  
        }  
    }  
}
```

Dacă în fișierul text exemplu.txt există cel puțin două numere scrise incorect (de exemplu: 2a17


```
finally {  
    System.out.println(s);  
}  
}  
}
```

Dacă în fișierul text exemplu.txt există cel puțin două numere scrise incorect (de exemplu: 2a17, a217, abc, 2-17 etc.), atunci programul: *

(5 Points)

- ☐ va afișa suma numerelor din fișier aflate înaintea primului număr scris incorect, după care programul se va termina forțat din cauza unei excepții
- ☒ va afișa o singură dată mesajul Număr incorect! și suma numerelor din fișier aflate înaintea primului număr scris incorect, după care programul se va termina normal
- ☐ va afișa câte un mesaj Număr incorect! pentru fiecare număr din fișier scris incorect, apoi va afișa suma tuturor numerelor din fișier scrise corect, după care programul se va termina normal
- ☐ va afișa doar mesajul Număr incorect!, după care programul se va termina forțat din cauza unei excepții

15. Se consideră următoarele instanțieri:

```
String sir_1 = "Examen";
```

```
String sir_2 = "Examen";
```

```
String sir_3 = new String("Examen");
```

```
String sir_4 = new String("Examen"); *
```

(5 Points)

- ☐ toate cele 4 obiecte au referințe diferite
- ☐ toate cele 4 obiecte au aceeași referință
- ☒ obiectele sir_1 și sir_2 au referințe egale, iar obiectele sir_3 și sir_4 au referințe diferite
- ☐ obiectele sir_1 și sir_2 au referințe egale, obiectele sir_3 și sir_4 au referințe egale, dar referința obiectelor sir_1 și sir_2 este diferită de referința obiectelor sir_3 și sir_4

16. Se consideră următoarea linie de cod

sii_1 și sii_2 este diferența de referință obiectelor sii_3 și sii_4

16. Se consideră următoarea linie de cod

```
Consumer<String> c = System.out::println;
```

Pentru a se afișa un șir de caractere folosind consumatorul c este necesară următoarea linie de cod: *

(5 Points)

- ☐ c.test(șir)
- ☒ c.accept(șir)
- ☐ c(șir)
- ☐ c.apply(șir)

17. Dacă un obiect este accesat printr-o referință de tipul unei interfețe pe care o implementează,

17. Dacă un obiect este accesat printr-o referință de tipul unei interfețe pe care o implementează, atunci: *
(5 Points)

- ☒ se pot accesa metodele implementate din interfață, alături de cele moștenite din clasa Object
- ☐ se pot accesa toate metodele publice încapsulate în clasă, cele din interfață, alături de cele moștenite din clasa Object
- ☐ un obiect nu poate fi referit printr-o referință de tipul unei interfețe pe care o implementează
- ☐ se pot accesa doar metodele implementate din interfață

18. Operația de inserare a unui obiect într-o colecție de tip HashSet presupune *

18. Operația de inserare a unui obiect într-o colecție de tip HashSet presupune *
(5 Points)

- ☐ apelul metodei insert()
- ☐ doar apelul metodei hashCode()
- ☒ apelul metodelor hashCode() și equals()
- ☐ doar apelul metodei equals()

19. Considerăm următorul program Java:

```
class Exceptie_1 extends Exception {}  
class Exceptie_2 extends Exceptie_1 {}
```


19. Considerăm următorul program Java:

```
class Exceptie_1 extends Exception {}  
class Exceptie_2 extends Exceptie_1 {}  
  
public class Test {  
    static void met() throws Exceptie_2 {...}  
  
    public static void main(String args[]) {  
        try {  
            met();  
        } catch (Exceptie_1 b) {  
            System.out.print("Exceptie 1! ");  
        } catch (Exceptie_2 d) {  
            System.out.print("Exceptie 2! ");  
        }  
    }  
} *
```

(5 Points)

- ☐ Exceptie 1!
- ☐ Exceptie 2!
- ☐ Exceptie 1! Exceptie 2!
- ☒ nimic, deoarece programul nu poate fi rulat din cauza erorii de compilare "Exception Exceptie_2 has already been caught"

● already been caught"

20. Mecanismul de supraîncărcare se poate realiza: *
(5 Points)

- ☒ doar între metode dintr-o singură clasă
- ☐ doar între o metodă dintr-o subclasă și o metodă din superclasa sa
- ☐ în limbajul Java nu se poate realiza mecanismul de supraîncărcare
- ☐ doar între metode publice