Test POO - versiunea A Nume ..... Grupa .....

A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a																				
b																				
c																				
d																				

1. Care dintre următoarele metode reprezintă variante corecte de supraîncarcare a `void act()`?
   1. `void act(int i)`
   2. `int act()`
   3. `void act(int i) throws Exception`
   4. `void act() throws Exception {throw new Exception();}`

   (a) 1, 2, 3, 4
   (b) 3, 4
   (c) 1, 3
   (d) 1, 2

2.
```
package pkg1;
public class Foo {
    public int a = 5;
    protected int b = 6;
    int c = 7;
}
---alt fisier:
package pkg2;
import pkg1.*;
public class Bar {
    public static void main(String[] args) {
        Foo f = new Foo();
        System.out.println(f.a);
        System.out.println(" " + f.b);
        System.out.println(" " + f.c);
   }
}
```

   (a) 5 6 7
   (b) programul are 2 erori de compilare
   (c) programul are o eroare de compilare
   (d) 5 urmat de o exceptie

3. Care este diferența între clase interne statice (CIS) și nestatice (CIN)?

   (a) CIN sunt vizibile doar în clasa exterioară acestora. CIS sunt vizibile oriunde în program
   (b) CIN pot fi instanțiate doar în clasa exterioară. CIS pot fi instanțiate oriunde în program
   (c) CIS nu pot fi moștenite, în schimb cele nestatice da
   (d) CIN nu pot fi instanțiate fără crearea, în prealabil, a unei instanțe a clasei exterioare

4.
```
class B {
        public static void foo() {
                System.out.println("1");
                this.foo();
        }
        private void foo(int x) {
                System.out.println("2");
        }
}
public class A extends B {
        public static void foo() {
                System.out.println("3");
        }
        public static void main(String[] args) {
                B.foo();
                A.foo();
        }
}
```

   (a) Eroare la linia `static void foo()` din clasa A
   (b) Se afișează 1 2 3 pe linii separate
   (c) Eroare la linia `public static void foo()` din clasa B
   (d) Eroare la linia `this.foo()`

5. Care dintre următoarele patternuri permite crearea de obiecte fără a expune logica lor clientilor?

   (a) Singleton
   (b) Observer
   (c) Factory
   (d) Visitor

6. Care dintre următoarele afirmații sunt adevarate?
   1. Un fișier sursă poate conține oricate definiții de clase, cel mult una putand fi marcată public.
   2. Un fișier sursă poate conține oricate clase marcate private.
   3. Un fișier sursă poate conține oricate interfețe, atât timp cât nu există și definiții de clase în respectivul fișier.
   4. In cadrul unui fișier sursă instrucțiunile "import" nu pot succeda definiția unei clase sau interfețe.

   (a) 1,3
   (b) 1,2
   (c) 1,2,3,4
   (d) 1,4

7. Care dintre următoarele interfețe nu extinde interfața *Collection*?

   (a) Set
   (b) Map
   (c) List
   (d) Queue

8. Cu ce poate fi înlocuită linia 7 pentru a obține o instanță a clasei B?

```
class A {
    class B {}
}
class Test {
    public static void main(String args[]) {
        A a = new A();
        /* Linia 7 */
    }
}
```

   (a) A.B b = new A.new B();
   (b) A.B b = new A().new B();
   (c) A.B b = new A.B();
   (d) B b = new B();

9. Câte linii sunt incorecte?

```
interface MyStruct<E> {
    static String name = "MyStruct";
    long ID;
    public void setElement(E e);
    abstract E getElement();
    protected void print();
}
```

   (a) 2
   (b) 0
   (c) 3
   (d) 1

10. Care dintre afirmațiile de mai jos caracterizează supraîncarcarea și suprascrierea ?
    A) Suprascrierea se referă la definirea, într-o clasa copil, de metode cu același nume, număr și tip de parametri, ca în clasa parinte;
    B) Supraîncarcarea se referă la definirea de metode cu același nume, însă tip de retur și parametri diferiți;
    C) Suprascrierea se referă la ascunderea implementării unor metode dintr-o clasa părinte, prin modificarea parametrilor acesteia;
    D) Supraîncarcarea se referă la definirea mai multor metode cu același nume, însa număr și/sau tip de parametri diferiți;

    (a) A și D
    (b) C și D

(c) C și B
(d) A și B

11.
```
class A {
        public int a = 1;
        float b = 2.0f;
        protected double c = 3.0d;
        private char d = 'x';
}
class B extends A { B () { c = a + b + d; }}
```

(a) programul nu va genera eroare la compilare
(b) eroare la compilare din cauză că modificatorii de acces ai variabilelor a, b, d sunt diferiți
(c) eroare la compilare din cauză că variabila d nu poate fi accesată în clasa B
(d) eroare la compilare din cauză că nu putem aduna o variabilă de tip char, cu una de tip float

12. Ce pattern se foloseste daca se vrea adaugarea facila de noi functionalitati in viitor, fara modificarea structurilor existente?

(a) Factory
(b) Singleton
(c) Visitor
(d) Pizza

13.
```
try {
    try {
        System.out.print("A");
        throw new Exception("1");
    } catch (Exception e) {
        System.out.print("B");
        throw new Exception("2");
    } finally {
        System.out.print("C");
        throw new Exception("3");
    }
} catch (Exception e) {
    System.out.print(e.getMessage());
}
```

(a) ABC3
(b) ABC
(c) AB2
(d) AB2C3

14.
```
public class Parent {
    public Parent(String s) { System.out.println("B"); }
    public static void main(String[] args) {
        System.out.println("C");
        Child c = new Child("C");
    }
}
class Child extends Parent {
    public Child(String s) { System.out.println("D"); }
}
```

(a) programul va afișa "DB"
(b) programul va afișa "BD"
(c) programul va afișa "C" urmat de o excepție
(d) eroare la compilare

15. Care este un apel corect al metodei `foo`?

```
class C { public void foo() { } }
class B { public static class D extends C { } }
```

(a) `new B.D.foo()`
(b) `new B.D().foo()`
(c) `B.D.foo()`
(d) `new B.D().super.foo()`

16. Care variantă este corectă pentru a completa TODO-ul?

```
class Foo{
    //TODO
    public void f(int n, int t){
        ArrayList<Float> y = new ArrayList<Float>();
        for (int i = 0; i < 10; i++)
            y.add(i * t * 1.0f);
        x.put(n, y);
    }
}
```

(a) `Map<Integer, ArrayList<Float>> x = new HashMap<Integer, ArrayList<Float>>();`
(b) `HashMap<Integer, ArrayList<Float>> x = new HashMap<Number, ArrayList<Number>>();`
(c) `Map<Integer, List<Float>> x = new Map<Integer, ArrayList<Float>>();`
(d) `HashMap<Integer, List<Float>> x = new HashMap<Integer, ArrayList<Float>>();`

17. Identificați afirmațiile corecte:
1. Un constructor nu poate avea un specificator de acces atașat definiției.
2. O clasă poate implementa mai multe interfețe, însă poate extinde o singură clasă abstractă.
3. Interfețele pot implementa alte interfețe.
4. După instanțierea unui vector new Object[10] toate elementele au valoarea null.

(a) 2,4
(b) 1, 2
(c) 1, 2, 3
(d) 1,2,4

18.
```
class Animal {
    Animal() { System.out.println("x "); }
}
class Horse extends Animal {
    Horse() { System.out.println("v "); }
    Horse(String race) {
        this();
        System.out.println("xf " + race);
    }
    public static void main(String[] args) {
        Horse h = new Horse("b ");
    }
}
```

(a) xf b
(b) x v xf b
(c) v xf b
(d) xf b v x

19. Care dintre următoarele implementări ale interfeței *Map* menține cheile sortate?

(a) HashMap
(b) LinkedHashMap
(c) TreeMap
(d) Hashtable

20.
```
public class Test {
      public static int hash(Object o){
            return o.hashCode();
      }
      public static void main(String args[]){
            try {
                  System.out.print("A");
                  int x = hash(null);
                  System.out.print("B");
            }
            catch(RuntimeException e){
                  System.out.print("C");
            }
            catch(Exception e){
                  System.out.print("D");
            }
            finally{
                  System.out.print("E");
            }
      }
}
```

(a) niciuna dintre variante
(b) ACE
(c) ADE
(d) ABE

A