2024.01.16 **R2**

```
abstract class Building {
    private int surface;
    public static int pricePerSquareMeter = 800;

    public Building(int surface) { this.surface = surface; }

    public abstract int getPrice();

    final void show2DPlan() { System.out.println("Show image with the
dwelling's 2D plan"); }
    @Override
    public String toString() { return "Building{" + "surface=" + surface + '}';
}
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Building building = (Building) o;
        return surface == building.surface;
    }

    @Override
    public int hashCode() { return 1; } }

class Apartment extends Building {
    protected enum Comfort {LOW, HIGH}
    private final Comfort comfort;
    public Apartment(int surface, Comfort comfort) {
        super(surface);
        this.comfort = comfort;
    }
    @Override
    public int getPrice() {
        int startPrice = pricePerSquareMeter * surface;
        if (comfort == Comfort.LOW) return startPrice;
        return startPrice * 2;
    }
    @Override
    void show2DPlan() { System.out.println("Display the apartment's 2D plan");
} }

class House extends Building {
    private final int yardArea;

    public House(int surface, int yardArea) {
        super(surface);
        this.yardArea = yardArea;
    }
    @Override
    public int getPrice() { return pricePerSquareMeter * surface + yardArea *
1000; }
    @Override
    void show2DPlan() { System.out.println("Display the house's 2D plan"); }
}
@FunctionalInterface
interface Filterable { boolean accept(Object o); }
```

1. Codul Java de mai sus apare în același fișier cu funcțiile statice de mai jos.
1.a Pentru codul de mai sus, precum și pentru cele două funcții de mai jos, determinați dacă există erori de compilare/execuție. În cazul erorilor de compilare, cum pot fi ele rezolvate? **[1p]**
1.b Odată ce ați rezolvat erorile de compilare, determinați și explicați în cuvintele voastre rezultatul execuției (ce se afișează pe ecran, sau cauza erorii la executarea programului) **[3p]**.

```java
public static void function2() {
    Filterable f = o -> {
        if (o instanceof String) {
            String s = (String) o;
            return s.toLowerCase().startsWith("s");
        } else if (o instanceof Building) {
            Building d = (Building) o;
            return d.getPrice() < 150000;
        }
        return false;
    };
    System.out.println(f.accept(new Apartment(70, Apartment.Comfort.HIGH)));
    System.out.println(f.accept(new House(100, 100)));
    List<String> strings = Arrays.asList("Seth", "sat", "in", "the", "sun",
"singing", "funny", "songs");
    strings.stream().filter(o -> (!f.accept(o))).forEach(System.out::println);}
```

```java
public static void function4() {
    Building.pricePerSquareMeter = 1;

    var buildings = new HashSet<Building>();
    buildings.add(new House(125, 100));
    buildings.add(new House(125, 500));
    buildings.add(new House(200, 250) {
        @Override
        public int getPrice() { return 100_000; }
    });
    System.out.println(buildings.size());
    // mapToInt returneaza un flux de intregi prin aplicarea functiei parametru
    // pe toate elementele fluxului
    var total = buildings.stream().mapToInt(Building::getPrice).sum();
    System.out.println("Total price: " + total); }
```

2. Se dă diagrama de clase UML de mai jos. Un obiect de tip **Permission** poate fi afișat pe ecran prin utilizarea metodei *toString*(). Aplicația permite crearea a două tipuri de obiecte de permisiune, **ReadPermission** (*toString()* returnează "*read permission*" și **WritePermission** (*toString()* returnează "*write permission*").
2.a Implementați clasele ***Permission***, ***ReadPermission*** și ***WritePermission*** **[1.5p]**
2.b Implementați clasele **FileSystemComponent**, **File** și **Directory**. Clasa **FileSystemComponent** este abstractă, are un nume și o permisiune. Funcțiile *showDetails() și add()* sunt abstracte. Fiecare **File** are o dimensiune (atributul *size*). Afișarea detaliilor unui **File** implică afișarea numelui, dimensiunii și a permisiunii sale. Un **Directory** poate conține mai multe **FileSystemComponent**. Acestea pot fi adăugate unui **Directory**. Afișarea detaliilor unui **Directory** implică afișarea detaliilor tuturor componentelor acelui **Directory [2p]**.
2.c Scrieți o funcție **main** care creează un fișier numit "Document.txt" de dimensiune 50 cu permisiune de citire; un **File** numit "Story.txt" de dimensiune 100 cu permisiune de citire; un **File** numit "Image.png" de dimensiune 200 cu permisiune de scriere; un **Directory** numit "Documents" care conține cele două fișiere document, cu permisiune de citire; un **Directory** "Images" care conține fișierul imagine, cu permisiune de scriere; un **Directory** "root" care conține cele două **Directory** create anterior, cu permisiune citire. Afișați detaliile pentru directorul "root" **[1.5p]**.