Java – Interface i klasy abstrakcyjne

Anna Gogolińska

Interface

- Słowo kluczowe ***Interface***.
  - Zawiera:
    - Nazwa
    - Zestaw metod bez ich implementacji (metody są zawsze *public* – nie trzeba tego zaznaczać)
    - Może również zawierać stałe (pola które są *final*, inicjalizowane bezpośrednio przy deklaracji)
  - Może dziedziczyć z wielu innych interfejsów.
- To szkielet/wymóg jaki muszą spełniać klasy go impementujące.

Implenetacja

- Klasy implementują interfejsy – słowo kluczowe *implements* w definicji klasy, później po przecinku wszystkie implementowane interfejsy.
- Klasa może implementować wiele interfejsów.
- Klasa musi implementować (zawierać) wszystkie metody, które są w implemenetowanych przez nią interfesjach.

```
public interface InterfaceA {
   public final int STALA = 5;
   public int methodA(int arg);
   public void methodB();
}

public class ClassA implements InterfaceA {
    @Override
    public void methodA(int arg){
       //jakiś kod
     }

    @Override
    public void methodB() {
       //jakiś kod
     }

    public void methodC() {
       //jakiś kod
 }
}
```

Klasa abstrakcyjna

- Słowo kluczowe ***abstract*** przed słowem *class* w definicji klasy.
- Nie można tworzyć obiektów takiej klasy – służy tylko aby z niej dziedziczyć.
- Może zawierać „normalne” metody wypełnione kodem oraz metody abstrakcyjne (*abstract* przed zwracanym typem), czyli takie jak w interfejsach, które muszą być implementowane przez klasy potomne.
- Może zawierać konstruktor, ale można go użyć tylko w klasach potomnych.

```
public abstract class ClassA{
   protected int valueOne;
   public void methodA(int arg) { //jakiś kod; }
   public abstract int methodB();
   public double methodC(int arg, int arg2) { //jakiś kod; }
}

public class ClassB extends ClassA {

   @Override
   public void methodA(int arg)
   {
        //jakiś kod
        super.methodA();
   }

   @Override
   public int methodB()
   {
        //jakiś kod
   }

   //ma też metodę methodC()
}
```

Klasy abstrakcyjne a interfejsy

- Obu używa się jeśli chce się wymóc jakie metody mają zawierać klasy (nazwy, typ wyniku, argumenty).
- Można dziedziczyć tylko z jednej klasy, ale implementować wiele interfejsów.
- Klasy abstrakcyjne pozwalają przygotować implementację części metod i/lub wyodrębnić wspólne fragmenty kodu.

Comparable

- Interfejs zdefiniowany w Javie. Zawiera metodę *compareTo(Object o)*, która zwraca liczbę całkowitą:
  - Ujemną jeśli obiekt na którym wywołujemy metodę jest mniejszy niż *o*, dodatnią jeśli większy, równą 0 jeśli taki sam.
  - Argument jest typu *Object* ale należy go zrzutować na typ klasy, która implementuje interfejs
  - Wewnątrz metody trzeba samemu zdefiniować jak porównywać obiekty i od czego ma zależeć wynik.
- Metoda ta (czyli implementacja interjefsu) wymagana jest jeśli chcemy sortować obiekty używając bibliotek javy (dla tablic: *Arrays.sort(tablica)*, dla list: *Collections.sort(lista)*).

Losowanie

- Tworzenie obiektu generatora liczb losowych:

```
Random generator = new Random();
```

- Klasa Random ma wiele metod losowania różnego typu wartości. Przykłady użycia:
  - `generator.nextInt()`
  - `generator.nextInt(n)`
  - `generator.nextDouble()`
- Sprawdzić w NetBeans dokumentację.

Polimorfizm

- Jeśli klasy dziedziczą z jednej klasy nadrzędnej lub impelementują jeden interfejs możemy zadeklarować typ obiektu jako klasa nadrzędna/interfejs, a podstawić pod niego obiekty klas potomnych/implementujących.
- Jest to szczególnie wygodne przy tablicach czy lista, można zadeklarować typ elementów w tablicy/liście jako klasę nadzędną lub interfejs i w tablicy/na liście umieszczać elementy wszystkich typów dziedziczących/implementujących.

Załóżmy, że mamy interfejs: InterfaceA, który implementując klasy: ClassA, ClassB i ClassC.

Załóżmy, że mamy klasę abstrakcyjną ClassAbs, z której dziedziaczą klasy: ClassZ, ClassY, ClassX.

Wtedy poprawne zapisy to:

```
InterfaceA obj = new ClassA();
InterfaceA obj2 = new ClassB();
obj = new ClassC();

ClassAbs obj3 = new ClassZ();
ClassAbs obj4 = new ClassX();

List<InterfaceA> lista = new ArrayList<>();
lista.add(new ClassA());
lista.add(obj);
lista.add(obj2);

ClassAbs[] tab = new ClassAbs[10];
tab[0] = new ClassY();
tab[1] = obj3;
tab[2] = obj4;
```