Typy generyczne

Typy generyczne (wprowadzone od 1.5) umożliwiają tworzenie elastycznego kodu. W zależności od przekazywanego parametru możemy zmieniać zachowanie obiektu. Zaleca się stosowanie następujących parametrów w zależności od przeznaczenia: E – element, K – Klucz, V- wartość, N – liczba, T,R,S – typ

Przykład. Zdefiniujemy teraz generyczną listę, która przyjmuje obiekty dziedziczące po klasie Number i zachowuje się jako ArrayLista.

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/generics/MyGenericList.java

```
public class MyGenericList<T extends Number> {
   private List<T> list;

   public void addElement(T t) {
      list.add(t);
   }

   public void removeElement(T t) {
      list.remove(t);
   }

   public MyGenericList() {
      list = new ArrayList<T>();
   }

   public List<T> getList() {
      return list;
   }
```

WildCard

Przykład. Napiszemy metodę, która sumuje liczby **dowolnego** typu (ale dziedziczące po klasie Number) zapisane na liście

https://github.com/idzikpro/JavaBasics/blob/master/src/main/java/pl/idzikpro/generics/GenericsListMain.java

```
public double sum(List<? extends Number> list){
   double sum=0;
   for (Number number:list
        ) {
      sum=sum+number.doubleValue();
   }
   return sum;
}
```

Znak **?** nazywany wildcard pozwala na podstawienie dowolnego typu, w tym przypadku takiego, który dziedziczy po klasie *Number*. Taki rodzaj ograniczenia nazywa się **upper bound.**

Użycie **super** w powyższym przykładzie oznaczałoby listę typów od Number „w górę” hierarchi dziedziczenia. Jest to nazywane **lower bound.**

Zalety

- kontrola poprawności typów już na poziomie kompilacji
- nie ma konieczności rzutowania i sprawdzania typów przy użyciu **instanceOf**