Java Generics

Typy generyczne

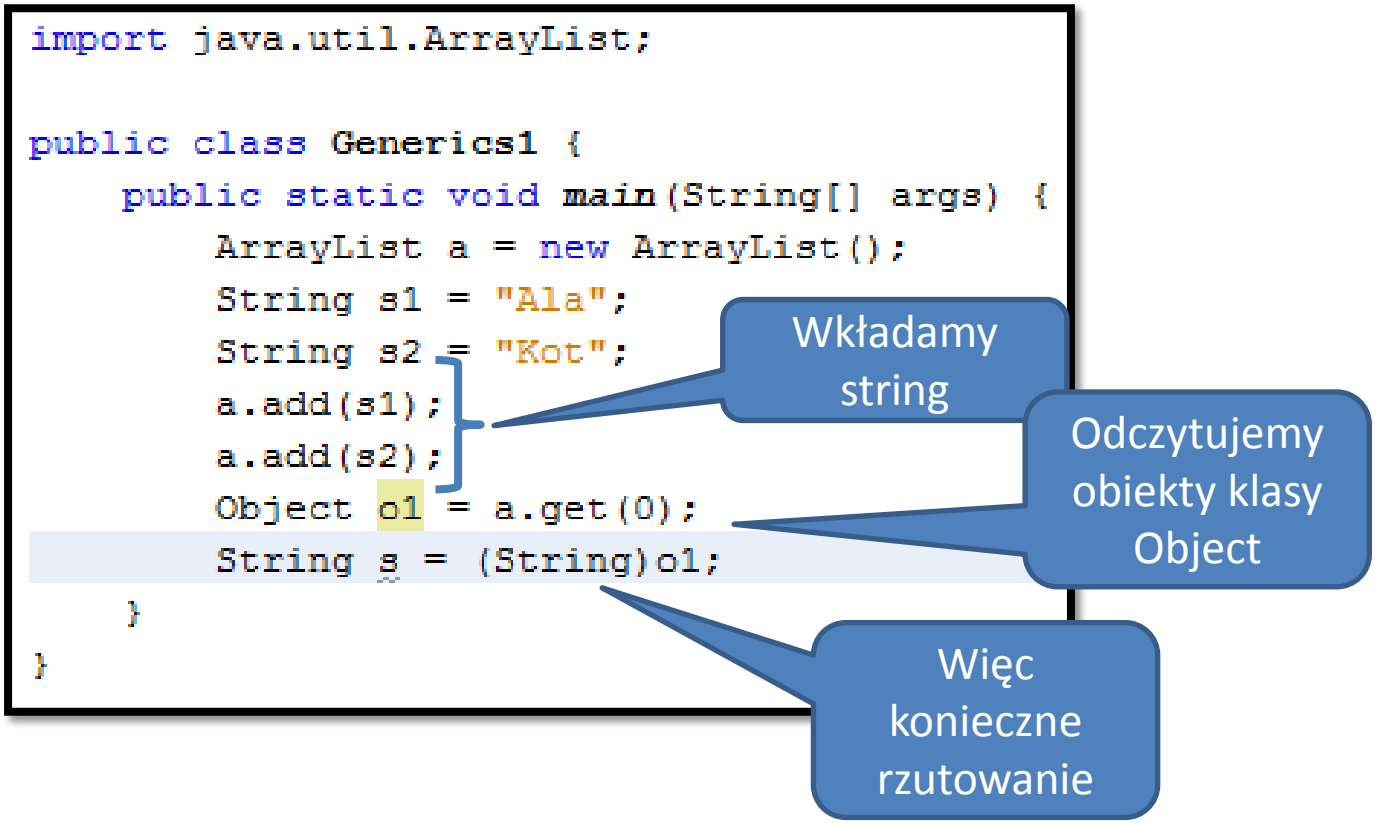
Java – język silnie typowany

Często tworzymy klasy, których działanie jest ogólne (niezależne od typu), ale chcemy aby wynik działania poszczególnych metod był określonego typu. Tzw. Typy parametryczne

Przykład: kolekcje

Jak nie ma typów generycznych

Tworząc dynamiczną tablicę: ArrayList()



Co jest bez typów generycznych

```
public class Generics1 {
    public static void main(String[] args) {
        ArrayList a = new ArrayList();
        String s1 = "Ala";
        String s2 = "Kot";
        Osoba o = new Osoba("Jan","Kowalski");
        a.add(s1);
        a.add(s2);
        a.add(o);
        Object w1 = a.get(0);
        String n1 = (String)w1;
        Object w2 = a.get(2);
        String n2 = (String)w2;
    }
}
```

Wkładamy do kolekcji dwa Stringi i osobę

Problem – nie wiemy co odczytujemy

Wyjątek, bo w2 to osoba a nie string więc ClassCastException

Typy generyczne

- Po co więc typy generyczne
  - Pozwalają zdefiniować typ (klasę) danych wejściowych i wyjściowych do/z metod.
  - Innymi słowy typ jest parametrem
  - Po co – nie chcemy wyjątków/błędów w trakcie „Run Time”, wolimy błędy kompilatora

- Typ wejściowy/wyjściowy definiowany po nazwie klasy w postaci < ***Nazwa*** > gdzie ***Nazwa*** – to typ danych

Typy generyczne

```
public class RejestrPierscieniowy<T> {
    Object[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;
    public RejestrPierscieniowy(int rozmiar){
        tablica = new Object[rozmiar];
    }

    public void dodaj(T t){
        tablica[licznikZapis] = t;
        licznikZapis++;
        licznikZapis %= tablica.length;
    }

    public T odczytaj(){
        Object o = tablica[licznikOdczyt];
        licznikOdczyt++;
        licznikOdczyt %= tablica.length;
        T out = (T)o;
        return out;
    }

    public static void main(String[] args) {...}
}
```

Symbol w miejsce którego podana zostanie nazwa klasy do przechowywania

Określamy typ (klasę) zmiennej „t”

Metoda zwraca wynik określonego typu

Aby odczytać właściwy wynik sami dokonujemy rzutowania

Typy generyczne - wykorzystanie

```
RejestrPierscieniowy<Integer> t = new RejestrPierscieniowy<>(10);
```

Od Java 7 notacja Diamentowa – mniej pisania

```
public static void main(String[] args) {
    RejestrPierscieniowy<Integer> t = new RejestrPierscieniowy<Integer>(10);
    for(int i=0; i<20; i++){
        t.dodaj(i);
    }
    for(int i=0; i<20; i++){
        int l = t.odczytaj();
        System.out.println(l);
    }
}
```

Przygotowujemy naszą klasę na operowanie na typach Integer

W pętli dodajemy objekty typu Integer

W pętli odczytujemy obiekty klasy Integer
Nie trzeba rzutowania

Dlaczego nie tak?

```
public class RejestrPierscieniowy<T> {
    T[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;

    public RejestrPierscieniowy(int rozmiar){
        tablica = new T[rozmiar];
    }

    public void dodaj(T t){
        tablica[licznikZapis] = t;
        licznikZapis++;
        licznikZapis %= tablica.length;
    }

    public T odczytaj(){
        T out = tablica[licznikOdczyt];
        licznikOdczyt++;
        licznikOdczyt %= tablica.length;
        return out;
    }
}
```

Dlaczego nie tak?

- Java – 100% kompatybilność wstecz
- Problem – dawniej brak Java Generics
- W Java każda klasa jest bytem autonomicznym, nie wie nic o innych klasach
- Nie można więc zrobić bytu autonomicznego który tworzy instancję określonej klasy której nie zna!!!

Rozwiązanie:

- Java Generics to rzutowania realizowane przez kompilator. W rzeczywistości wszystko jest klasy Object, a kompilator niejawnie umieszcza operatory rzutowania (Tak jak my)

Gdzie więc jest błąd?

```
public class RejestrPierscieniowy<T> {
    T[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;

    public RejestrPierscieniowy(int rozmiar){
        tablica = new T[rozmiar];
    }
```

Bo „**T**” to symbol używany do rzutowania, a nie klasa, nie można więc stworzyć instancji w ten sposób

Więc może tak?

```
public class RejestrPierscieniowy<T> {
    T[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;
    public RejestrPierscieniowy(int rozmiar){
        Object[] o = new Object[rozmiar];
        tablica = (T[])o;
    }
```

Tworzymy tablicę objektów

Rzutujemy tablicę obiektów na tablicę T[]

Więc może tak?

Czy tak jest dobrze, czy może jest tu błąd?

```
public class RejestrPierscieniowy<T> {
    T[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;
    public RejestrPierscieniowy(int rozmiar){
        Object[] o = new Object[rozmiar];
        tablica = (T[])o;
    }
```

Tworzymy tablicę objektów

Rzutujemy tablicę obiektów na tablicę T[]

Więc może tak?

```
public class RejestrPierscieniowy<T> {
    T[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;
    public RejestrPierscieniowy(int rozmiar){
        Object[] o = new Object[rozmiar];
        tablica = (T[])o;
    }
```

ClassCastException, bo mamy tablicę Object a rzutujemy ją na tablicę T[]

Co jeszcze daje nam Java Generics

Ograniczenia na typy

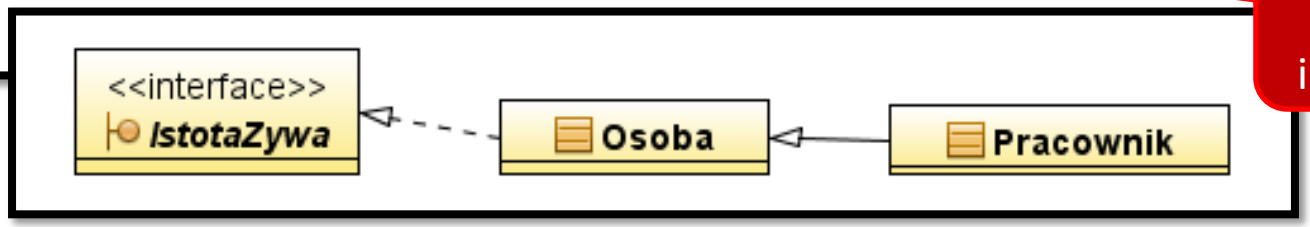
```
public class RejestrPierscieniowy<T extends Osoba> {
    Object[] tablica;
    int licznikZapis = 0;
    int licznikOdczyt = 0;
    public RejestrPierscieniowy(int rozmiar){...}
    public void dodaj(T t){...}
    public T odczytaj(){...}
    public static void main(String[] args) {
        RejestrPierscieniowy<Osoba> t = new RejestrPierscieniowy<>(10);
        Osoba o1 = new Osoba("Jan","Kowalski");
        t.dodaj(o1);
        Pracownik p1 = new Pracownik("Anna", "K");
        t.dodaj(p1);

        IstotaZywa iz = o1;
        t.dodaj(iz);
    }
```

Definiujemy że T musi być klasą pochodną od Osoba

Możemy dodać Osobę

Możemy dodać Pracownika, bo dziedziczy po osobie

Nie możemy dodać IstotyZywej bo jest interfejsem do Osoby



Problemy z Java Generics

- Wymaganie: typ generyczny musi być nazwą klasy!!!
- Problem: co z typami prymitywnymi jak boolean, int, double etc.
- Konieczne jest opakowanie:
  - Klasa Number i pochodne
    - BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short
  - Problem: wydajność

Przykład własnej paczki

```
public class Liczba<T extends Number> {
    final private T liczbaa;

    public Liczba(T l){
        this.liczbaa = l;
    }

    public T odczytaj(){
        return liczbaa;
    }

    public static void main(String[] args) {...}
}
```