Curs 5 PPOO

Prof. univ. dr. Cristian CIUREA

Departamentul de Informatică și Cibernetică Economică

cristian.ciurea@ie.ase.ro

Java fundamentals

- Genericitate în Java

Generics

- **Generics** permite unui tip de date sau unei metode să opereze pe obiecte de diferite tipuri, oferind în același timp siguranță la momentul compilării.
- Conceptul de **Generics** a fost încorporat în Java începând cu versiunea de JDK 5.0, împreună cu conceptul de **wildcards**.
- Se aseamănă cu conceptul de **template** din C++.

Generics

- **Generics** forțează siguranța la momentul compilării pe colecții sau alte clase și metode declarate utilizând parametrii de tipuri generice (verificarea tipului de date se face la compilare).
- Se utilizează sintaxa cu **<denumire tip>**.
- Se utilizează în majoritatea situațiilor pe colecții de obiecte.

Generics

```
public class Box {

Object value; //generic reference

public void setValue(Object value)    {this.value = value;}
public Object getValue()    {return value; }
}
```

A generic class – classic approach

VS.

```
public class GenericBox<T> {
T value; //generic type
public void setValue(T value) { this.value = value;}
public T getValue()    {return value;}
}
```

Generics

Convenții de denumire a tipurilor:

- **E - Element** (folosit în special de Java Collections Framework)
- **K - Key**
- **N - Number**
- **T - Type**
- **V - Value**

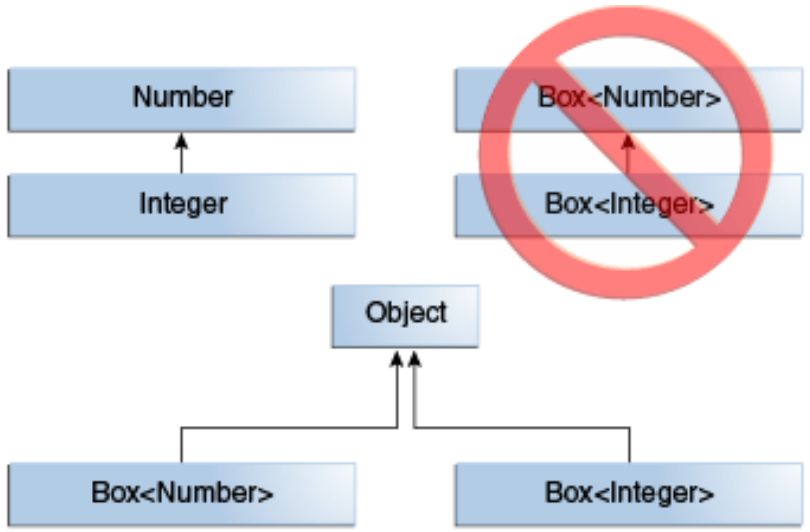
Generics

- Mecanismul genericității oferă un mijloc de abstractizare a tipurilor de date și este util mai ales în ierarhia de **colecții**.
- ```
  List lista= new LinkedList();
  lista.add(new Integer(0));
  Integer x = (Integer) lista.iterator().next();
  ```
- Se observă necesitatea operației de cast pentru a identifica corect variabila obținută din listă.
- Această situație are mai multe dezavantaje:
  - este îngreunată citirea codului;
  - apare posibilitatea unor erori la execuție, în momentul în care în listă se introduce un obiect care nu este de tipul *Integer*.
  ```

Generics

- Genericitatea intervine tocmai pentru a elimina aceste probleme.
- ```
  List<Integer> lista= new LinkedList<Integer>();
  lista.add(new Integer(0));
  Integer x = lista.iterator().next();
  ```
- În această situație, lista nu mai conține obiecte oarecare, ci poate conține doar obiecte de tipul *Integer*.
- Beneficiile dobândite prin utilizarea genericității constau în:
  - îmbunătățirea lizibilității codului;
  - creșterea gradului de robustețe.
  ```

Generics

- În ceea ce privește relația de moștenire, în cazul a două tipuri de date A și B (de exemplu, ***Number*** și ***Integer***), indiferent dacă A sau B sunt în relație de moștenire sau nu, ***MyClass<A>*** nu este în nici o relație cu ***MyClass<B>***.
- Părintele comun al claselor ***MyClass<A>*** și ***MyClass<B>*** este ***Object***.
- **`Integer`** `extends` **`Object`**
- **`Integer`** `extends` **`Number`**
- ~~**`Stack<Integer> extends Stack<Object>`**~~
- ~~**`Stack<Integer> extends Stack<Number>`**~~

Generics



Box<Integer> nu este subtip al lui ***Box<Number>*** chiar dacă ***Integer*** este subtip al clasei ***Number***.

Generics

- Java permite utilizarea de metacaractere pentru a servi ca argumente de tip pentru tipurile parametrizate. **Wildcard-urile** sunt argumente de tip de forma "?".
- Wildcard-urile sunt utilizate atunci când dorim să întrebuințăm o structură generică drept parametru într-o funcție și nu dorim să limităm tipul de date din colecția respectivă.

Generics

- Sintaxa unui wildcard permite unei metode generice să accepte subtipuri sau supertipuri ale tipului declarat.
- Wildcard-ul de forma **"?"** se numește **wildcard nemărginit** și arată că orice tip de instanțiere este posibilă.
- Ca un exemplu de wildcard nemărginit, **List <?>** indică o listă care are un tip de obiect necunoscut. Metodele care primesc o astfel de listă ca parametru vor accepta orice tip de obiect ca argument.

```
List<?> anyList = new ArrayList<Date>( );
anyList = new ArrayList<String>( );
```

Generics

- Sintaxa unui **wildcard mărginit** utilizează cuvintele cheie **extends** sau **super** pentru a limita domeniul de tipuri atribuite.

```
List<? extends Date> dateList = new ArrayList<Date>( );
dateList = new ArrayList<MyDate>( );
```

Generics

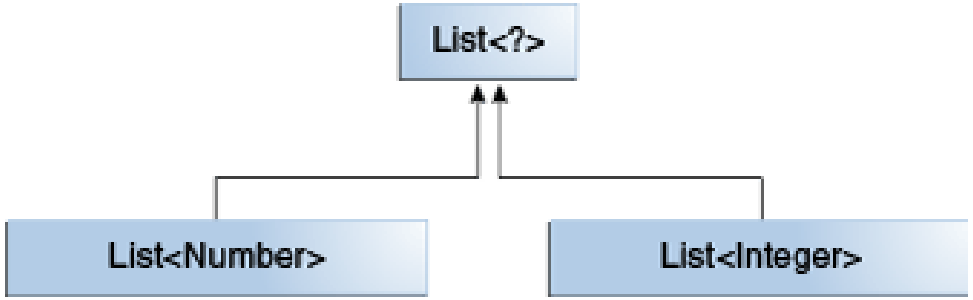
Wildcard-urile pot fi:

- delimitate superior: **List<? extends Number> list**
- delimitate inferior: **List<? super Integer> list**
- fără delimitare: **List<?> list**

Generics

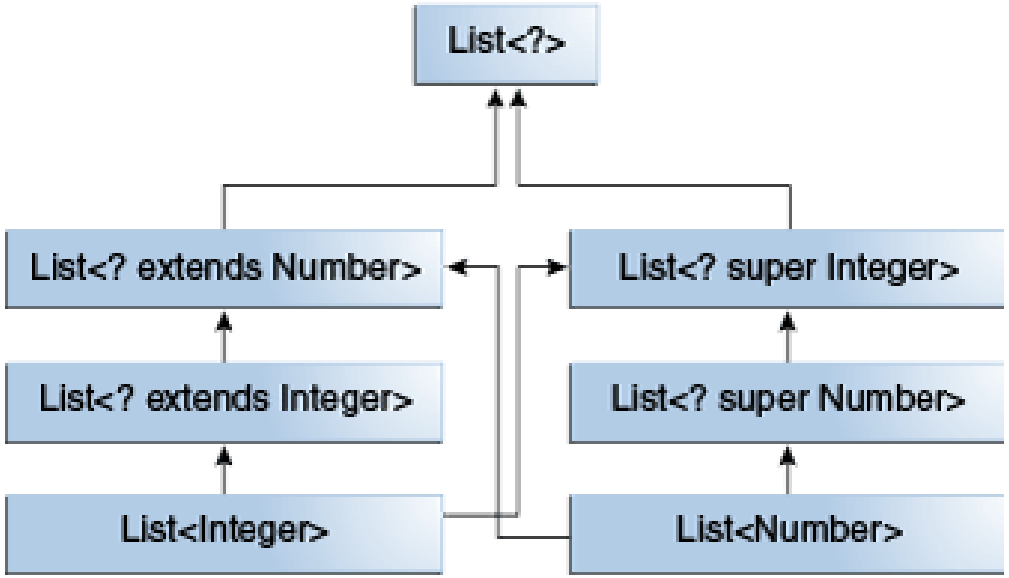
- De reținut acronimul ***PECS*** (**Producer Extends, Consumer Super**).
- **Producer Extends:** dacă avem nevoie de o colecție List pentru a produce valori de tip **T** (dorim să citim obiecte de tip T din listă), trebuie să o declarăm **<? extends T>**, dar nu putem adăuga elemente la această listă.
- **Consumer Super:** dacă avem nevoie de o colecție List pentru a consuma valori de tip **T** (dorim să scriem obiecte de tip T în listă), trebuie să o declarăm **<? super T>**, dar nu există garanții referitor la ce tip de elemente putem citi din această listă.

Generics



Chiar dacă ***Integer*** este subtip al lui ***Number***, ***List<Integer>*** nu este subtip al lui ***List<Number>*** și, de fapt, aceste două tipuri nu sunt înrudite.
Părintele comun al lui ***List<Number>*** și ***List<Integer>*** este ***List<?>***.

Generics



Ierarhia claselor generice ***List<?>***

Generics

Clase generice:

- Sunt clase template similar ca în C++, adică sunt șabloane de clase cu parametri;
- Pot fi adaptate pe tipuri reale, concrete (tipurile fundamentale din Java plus cele definite de utilizator);
- Prin instanțiere, JVM generează clase concrete;
- O clasă generică necesită unul sau mai multe tipuri parametrizate.

Generics

```
public class TestGenerics<T> {
        T instanceVariable;
        T[] array
        TestGenerics(T input){
            instanceVariable = input;
        }
        T getInstance(){
            return instanceVariable;
}

TestGenerics<int> ref = new TestGenerics<int>();
```

type variable

Generics

Metode generice:

- Sunt metode care introduc parametri proprii, alții decât cei ai clasei din care fac parte;
- Conțin declararea tipului parametrizat folosind sintaxa <> (operatorul ***diamond***);
- Permit creșterea gradului de generalizare prin definirea de șabloane de funcții;
- Sintaxa cu <> se pune înaintea tipului returnat al metodei;
- Spre deosebire de clasele generice, metodele generice nu trebuie să fie instanțiate cu un tip concret înainte de a fi utilizate.

```
<T> return_type method(parameters)
ex: <T> T doSomething(int val1, T val2){}
```

Bibliografie

- [1] Jonathan Knudsen, Patrick Niemeyer – *Learning Java, 3rd Edition*, O'Reilly.
- [2] http://www.itcsolutions.eu
- [3] http://www.acs.ase.ro
- [4] http://docs.oracle.com/javase/tutorial/ /index.html
- [5] http://cursuri.cs.pub.ro/~poo/wiki/index.php/Genericitatea
- [6] https://profs.info.uaic.ro/~acf/java/slides/ro/colectii_slide.pdf