Generics

Adding Type Safety and Code Reusability

SoftUni Team

Technical Trainers

Table of Content

2

SoftUni
Foundation

sli.do

#java-advanced

Generics

The Problem, The Solution

The Problem Before Java 5.0

- We need a collection that will store **only strings**

```
List strings = new ArrayList();

strings.add("1");

strings.add("2");

strings.add(3); // Is this correct?

String e1 = (String) strings.get(0);

String e2 = (String) strings.get(1);

String e3 = (String) strings.get(2); // RTE
```

Generics - Type Safety

- We need a collection that will store **only strings**

```
List<String> strings = new ArrayList<String>();

strings.add("1");

strings.add("2");

strings.add(3); // Compile time error
```

- Adds **type safety** and provides powerful way for **code reuse**

```
List<Integer> integers = new ArrayList<>();

List<Person> people = new ArrayList<>();
```

Type Inference

6

Generic Classes

- Defined with <**Type Parameter 1**, **Type Parameter 2** ... etc.>

```
class ArrayList<T> {

    /* voodoo magic */

}
```

- **Multiple** Type Parameters

```
class HashMap<K, V> {

    /* voodoo magic */

}
```

- You can use it anywhere inside the **declaring class**

```
class List<T> {
    public void add (T element) {…}
    public T remove () {…}
    public T get(int index) {…}
}
```

Problem: Jar of T

- Create a class **Jar<>** that can store **anything**

- Adding should add **on top** of its contents

- Remove should get the **topmost** element

- It should have two public methods:

  - **void add(element)**

  - **element remove()**

Solution: Jar of T

```java
public class Jar<T> {

  private Deque<T> content;

  public Jar() { this.content = new ArrayDeque<>(); }


  public void add(T entity) {

    this.content.push(entity);

  }


  public T remove() { return this.content.pop(); }
}
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

Subclassing Generic Classes

- Can **extend** to a concrete class

```java
class JarOfPickles extends Jar<Pickle> {

    …

}
```

```java
JarOfPickles jar = new JarOfPickles();

jar.add(new Pickle());

jar.add(new Vegetable()); // Error
```

Generic Interfaces

- **Generic interfaces** are similar to **generic classes**

```
interface List<T> {

    void add (T element);

    T get (int index);

    …

}
```

```
class MyList implements List<MyClass> {…}
```

```
class MyList<T> implements List<T> {…}
```

Problem: Generic Array Creator

- Create a class **ArrayCreator** with a single method:
  - **static T[] create(int length, T item)**
- Add a single overload:
  - **static T[] create(Class<T> class, int length, T item)**
- It should **return an array**
  - with the given length
  - every element should be **set to the given default item**

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

13

Solution: Generic Array Creator (1)

```java
public static <T> T[] create(int length, T item) {
    T[] array = (T[]) new Object[length];
    for (int i = 0; i < array.length; i++) {
        array[i] = item;
    }
    return array;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

Solution: Generic Array Creator (2)

```java
public static <T> T[] create(
                Class<T> cl, int length, T item) {
  T[] array = (T[]) Array.newInstance(cl, length);
  for (int i = 0; i < array.length; i++) {
    array[i] = item;
  }
  return array;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

Type Erasure

■ Generics are **compile time illusion**

```
List<String> strings = new ArrayList<String>();

System.out.println(strings instanceof List);


System.out.println(

    strings instanceof List<String>); // CTE
```

■ Compiler **deletes** all angle bracket syntax

■ Adds **type casts** for us (presented in byte-code)

```java
public class Illusion<T> {

  public void function(Object obj) {

    if (obj instanceof T) {} // Error

    T[] array = new T[1]; // Error

    T newInstance = new T(); // Error

    Class cl = T.class; // Error

  }

}
```

Type Parameter Bounds

Upper and Lower Bounds

Type Parameter Bounds

- **`<T extends Class>`** - specifies an **"Upper bound"**

```
class AnimalList<T extends Animal> {
    private List<T> animals;
    void add (T animal) {…}
    void putAnimalsToSleep() {
        for (Animal a : this.animals)
            a.sleep();
    }
}
```

T will be a subclass of Animal

Uses methods of **T**

Problem: Generic Scale

- Create a class **Scale<T>** that:

  - Holds two elements: **left** and **right**

  - Receives the elements through its single constructor:

    - Scale(**T** left, **T** right)

  - Has a method: **T getHeavier()**

- The greater of the two elements is heavier

- Should return **null** if the elements are equal

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

Solution: Generic Scale (1)

```java
public class Scale<T extends Comparable<T>> {

    private T left;

    private T right;

    public Scale(T left, T right) {

        this.left = left;

        this.right = right;

    }

    public T getHeavier() { /* next slide */ }    }
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

```
public T getHeavier() {

    if (this.left.compareTo(this.right) == 0)

        return null;

    if (this.left.compareTo(this.right) < 0)

        return right;

    return left;

}
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab

Problem: List Utilities

- Create a class **ListUtils** that:

    - Has two static methods:

        - **T getMin(List<T> list)**

        - **T getMax(List<T> list)**

    - Should throw **IllegalArgumentException** if an empty list is passed

Solution: List Utilities

```java
public static <T extends Comparable<T>> T getMax(List<T> list) {
    if (list.size() == 0) throw new IllegalArgumentException();
    T max = list.get(0);
    for (int i = 1; i < list.size(); i++) {
        if (max.compareTo(list.get(i)) < 0)
            max = list.get(i);
    }
    return max;
}
```

Check your solution here: https://judge.softuni.bg/Contests/1526/Generics-Lab
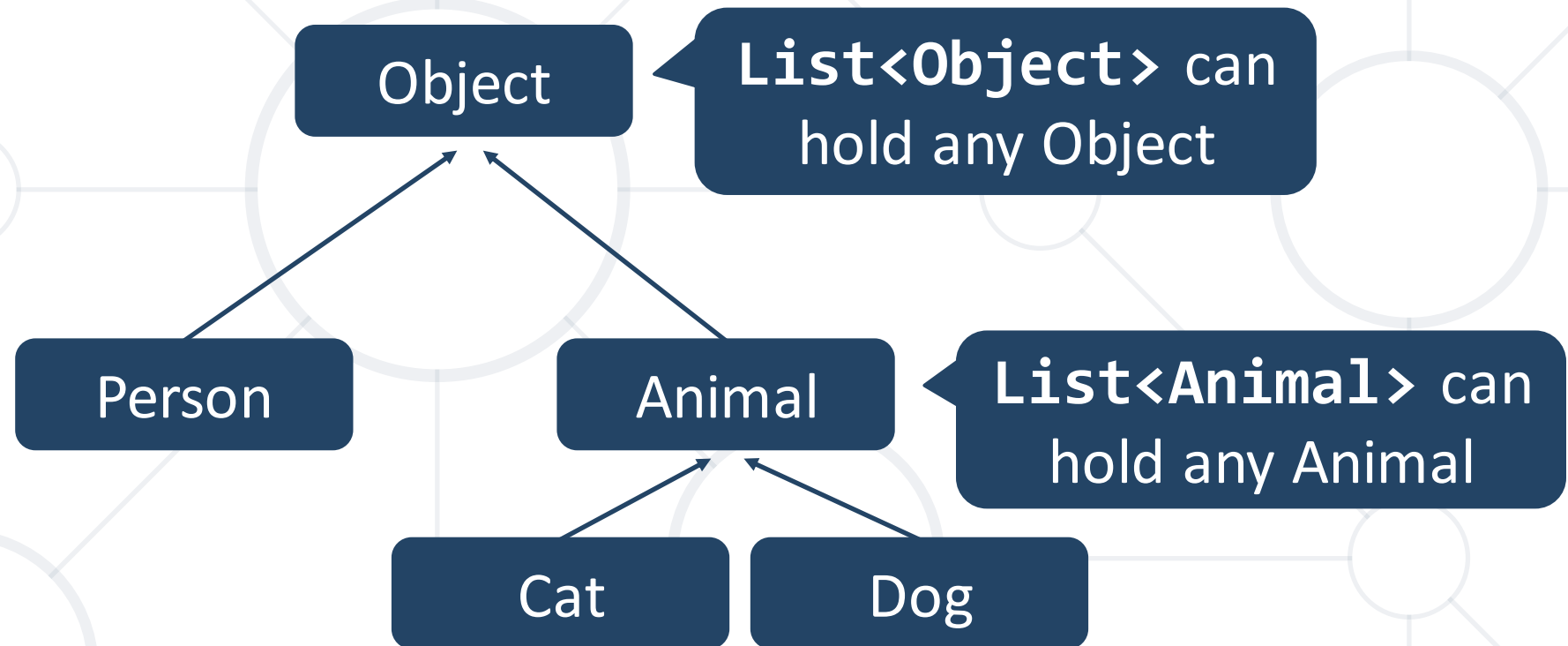
- Generics are **invariant**

```
List<Object> objects = new ArrayList<>();
List<Animal> animals = new ArrayList<>();
objects = animals; // Compile Time Error!
```

- If the above was possible, then why not:

```
objects = animals;
objects.add(new Person()); // Impossible!
```

Summary
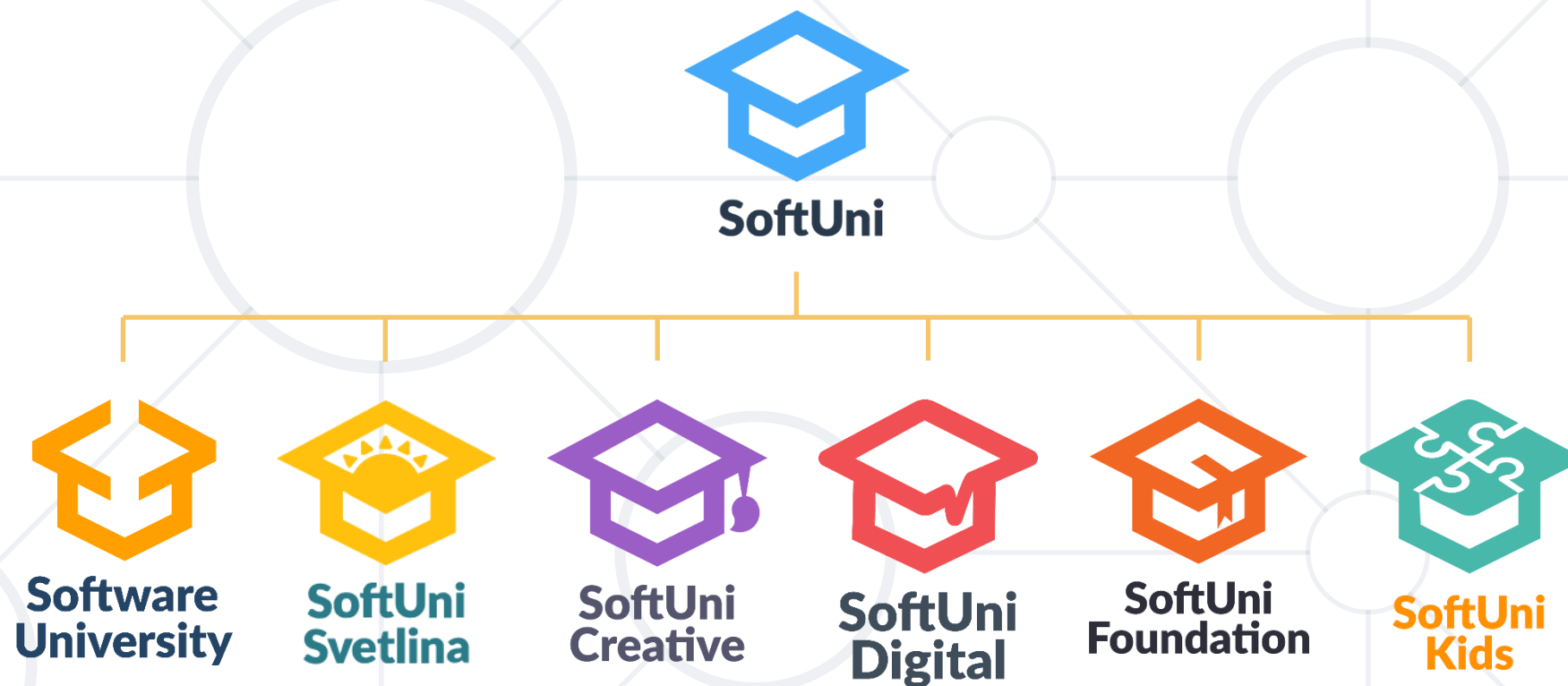
- Generics add **type safety**
- Generic code is **more reusable**
- **Classes**, **interfaces and methods can be generic**
- Runtime information about **type parameters** is lost due to **erasure**

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

https://softuni.bg/modules/59/java-advanced

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

STEMO
Computer Systems & Software

M

SoftUni Organizational Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license