Generics

Adding Type Safety and Code Reusability

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#java-advanced

Table of Contents

Generics

The Problem and 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

Software University

- 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 a powerful way for **code reuse**

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

Type Inference

Generic Classes

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

```
class ArrayList<T> {
  /* magic */
}
```

- **Multiple** Type Parameters

```
class HashMap<K, V> {
  /* magic */
}
```

Type Parameter Scope

- 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()`

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

Solution: Jar of T

```
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

```
class JarOfPickles extends Jar<Pickle> {
    …
}
```

```
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

Solution: Generic Array Creator (1)

```
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)

```
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

Software University

- 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)

Type Erasure – Example

```
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
  }
}
```

Wildcard

What is Wildcard?

- A question mark (**?**) is a **wildcard** in programming, representing an **unknown** type
- It is used as a parameter type, local variable, field, and less commonly as a return type
- Different instances of a generic type are not compatible with each other, but this changes when the wildcard is used as an actual type parameter

Upper Bounded Wildcards

- Given method works with List <Integer>, List <Double>, and List<Number>, you can replace the types using an upper bound wildcard
- The declaration is made using the "**?**" followed by the "**extends**" keyword followed by its **upper bound**.
  - `public static void add(List<? extends Number> list)`

Lower Bounded and Unbounded Wildcards

Software University

- The Lower Bound declaration is made using the "**?**" followed by the "**super**" keyword followed by its **lower bound**
  - `public static void add(List<? super Number> list)`
- The Unbounded Wildcard is used for a list of unknown data types and in the cases where:
  - has a method that can be invoked using functionality from the object class
  - code uses methods in the generic class that do not depend on the parameter type
  - `public static void add(List<?> list)`

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)

```
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

Solution: Generic Scale (2)

```
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

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

Solution: List Utilities

```
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

Type Parameters Relationships (1)

Software University

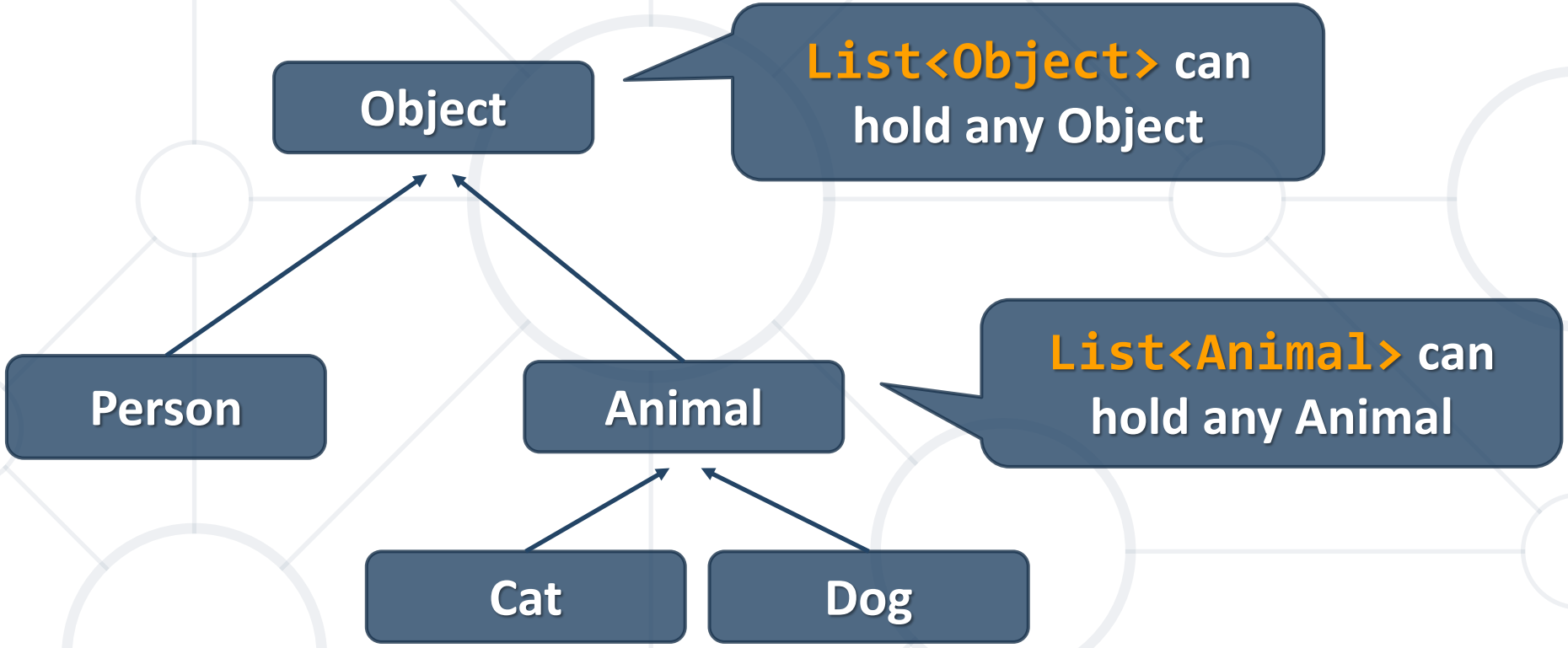
- Generics are **invariant**

```
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!
```

Type Parameters Relationships (2)

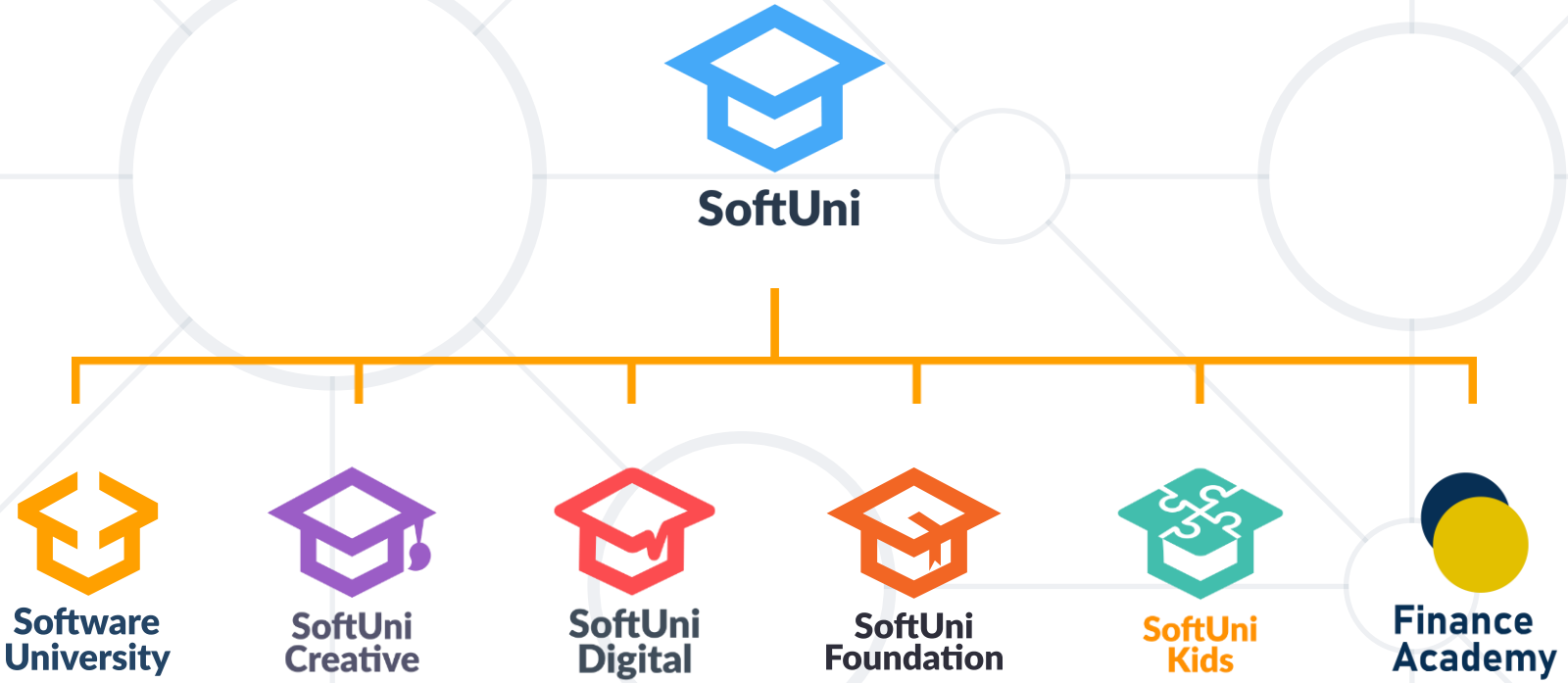



```
List<Object> ≠ List<Animal>
```

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 Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

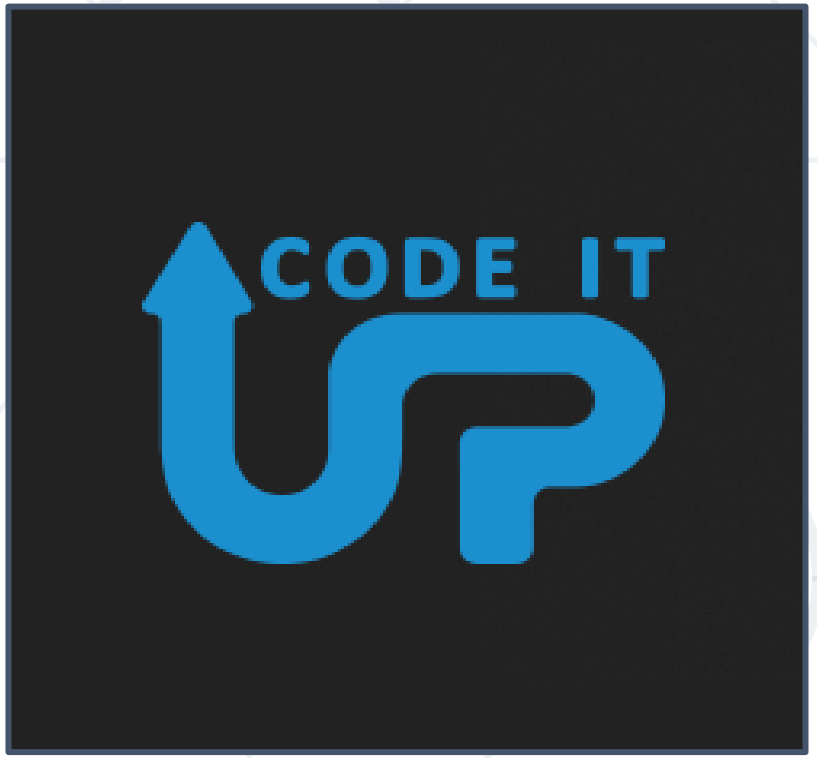
BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Software University



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg