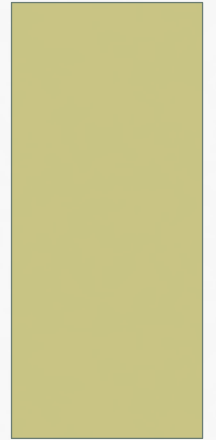


# Generics

Java



# МОТИВАЦИЯ

Java Generics

# ХРАНИЛИЩЕ ЭЛЕМЕНТОВ

```
public class Example01 {  
    public static void main(String[] args) {  
        ArrayList test = new ArrayList();  
  
        test.add("test");  
        test.add(3);  
        test.add(3.0);  
        test.add((Object)2);  
  
        for (int i = 0; i < test.size(); i++)  
            System.out.println(test.get(i));  
    }  
}
```

# ХРАНИЛИЩЕ ЭЛЕМЕНТОВ

```
public class Example01 {  
    public static void main(String[] args) {  
        ArrayList test = new ArrayList();  
  
        test.add("test");  
        test.add(3);  
        test.add(3.0);  
        test.add((Object)2);  
  
        for (int i = 0; i < test.size(); i++)  
            System.out.println(test.get(i));  
    }  
}
```

**Результат:**  
test  
3  
3.0  
2

# ХРАНИЛИЩЕ ЭЛЕМЕНТОВ

```
public class Example01 {  
    public static void main(String[] args) {  
        ArrayList test = new ArrayList();  
  
        test.add("test");  
        test.add(3);  
        test.add(3.0);  
        test.add((Object)2);  
  
        for (int i = 0; i < test.size(); i++)  
            System.out.println((String)test.get(i));  
    }  
}
```

# ХРАНИЛИЩЕ ЭЛЕМЕНТОВ

```
public class Example01 {  
    public static void main(String[] args) {  
        ArrayList test = new ArrayList();
```

Результат:

test

Exception in thread "main" java.lang.ClassCastException:  
java.lang.Integer cannot be cast to java.lang.String  
at Example01.main(Example01.java:14)

```
        for (int i = 0; i < test.size(); i++)  
            System.out.println((String)test.get(i));  
    }  
}
```

# ХРАНИЛИЩЕ ЭЛЕМЕНТОВ

```
public class Example02 {  
    public static void main(String[] args) {  
        ArrayList<String> test = new ArrayList<String>();  
  
        test.add("test");  
        test.add(3);  
        test.add(3.0);  
        test.add((Object)2);  
  
        for (int i = 0; i < test.size(); i++)  
            System.out.println(test.get(i));  
    }  
}
```

## Ошибка компиляции:

The method `add(String)` in the type `ArrayList<String>` is not applicable for the arguments `(double)`

# ВНУТРИ ХРАНИЛИЩА

```
class Holder1 {  
    private Automobile a;  
    public Holder1(Automobile a) { this.a = a; }  
    Automobile get() { return a; }  
}
```

Этот контейнер не универсален – позволяет хранить только **Automobile**



# ВНУТРИ ХРАНИЛИЩА-2

```
class Holder2 {  
    private Object a;  
    public Holder2(Object a) { this.a = a; }  
    public void set(Object a) { this.a = a; }  
    public Object get() { return a; }  
}
```

**Этот контейнер универсален – но не удобен в использовании и нет защиты от неверного типа данных**

```
Holder2 h2 = new Holder2(new Automobile());  
Automobile a2 = (Automobile)h2.get();  
h2.set("Not an Automobile");  
String s = (String)h2.get();  
h2.set(1); // Автоматически упаковывается в Integer  
Integer x = (Integer)h2.get();
```

**Именно так были реализованы контейнеры до Java 5**

# ВНУТРИ ХРАНИЛИЩА-3

```
class Holder3<T> {  
    private T a;  
    public Holder3(T a) { this.a = a; }  
    public void set(T a) { this.a = a; }  
    public T get() { return a; }  
}  
  
public class Example04 {  
    public static void main(String[] args) {  
        Holder3<Automobile> h3 = new Holder3<Automobile>(new  
Automobile());  
        Automobile a = h3.get(); // Преобразование не требуется  
        h3.set("Not an Automobile");// Ошибка  
        h3.set(1); // Ошибка  
    }  
}
```

**Один класс на любой тип объекта + не требуется преобразование + защита от неверного типа данных**

# GENERICIS

Java Generics

# КАК УСТРОЕНО

- В C++ шаблоны компилируются в новый класс при использовании каждого нового типа параметра. В итоге имеем **отдельный машинный код для каждого типа параметра**.
- В Java в скомпилированном байт-коде нет никакой информации о типе параметра. **Один байт-код для всех типов параметра**.
- При компиляции вся информация о типе параметра стирается и **превращается в наиболее общий тип** – Object, если нет дополнительных ограничений.

# КАК РАБОТАЕТ (на примере ArrayList)

- В байт-коде класса ArrayList написано, что он внутри себя хранит массив `Object[]` (а не `T[]`)

- Данный код:

```
ArrayList<String> a1 = new ArrayList<String>();  
a1.add("Мама");  
String tmp = a1.get(0);
```

- Преобразуется в:

```
ArrayList a1 = new ArrayList();  
a1.add((String)"Мама");  
String tmp = (String)a1.get(0);
```

# КАК РАБОТАЕТ

- Таким образом:
  - generic 'и позволяют работать единым образом с переменными различных типов
  - generic-код при компиляции теряет всю информацию о типе
  - по своей сути generic-класс – это обертка над классом с Object, которая проверяет (приводит) тип во всех точках входа и выхода из generic-кода. Тем самым проверяется корректность присваиваний и операций **на этапе компиляции**
  - для проверки корректности на этапе выполнения существуют, например, специальные коллекции.

# НЕСОВМЕСТИМОСТЬ GENERIC-ТИПОВ

- Generic-типы не совместимы по присваиванию

```
List<Integer> li = new ArrayList<Integer>();  
List<Object> lo = li;
```

- Иначе — ошибки

```
lo.add("hello"); // ClassCastException  
Integer li = lo.get(0);
```

# ПРЕОБРАЗОВАНИЕ ТИПОВ

- Уничтожение информации о типе
  - `List l = new ArrayList<String>();`
- Добавление информации о типе
  - `List<String> l = (List<String>) new ArrayList();`
  - `List<String> l1 = new ArrayList();`
- Unchecked warning
  - Ответственность программиста
  - `SuppressWarnings("unchecked")`



# ОГРАНИЧЕНИЯ GENERIC

- Невозможно создать массив параметра типа
  - `Collection<T> c;`
  - `T[] ta;`
  - `new T[10];`
- Невозможно создать массив Generic-классов
  - `new ArrayList<List<Integer>>();`
  - `List<?>[] la = new List<?>[10];`

Но этот код работает (правда с `unchecked`):

- `List<String>[] la = new List[10];`

# ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

Java Generics

# ПРОБЛЕМА 1

- Метод

```
void dump(Collection<Object> c) {  
    for (Iterator<Object> i = c.iterator(); i.hasNext(); ) {  
        Object o = i.next();  
        System.out.println(o);  
    }  
}
```

- ВЫЗОВЫ

- List<Object> l; dump(l);
- List<Integer> l; dump(l);

# РЕШЕНИЕ 1 – WILDCARD

- Метод

```
void dump(Collection<?> c) {  
    for (Iterator<?> i = c.iterator(); i.hasNext(); ) {  
        Object o = i.next();  
        System.out.println(o);  
    }  
}
```

- ВЫЗОВЫ

- List<Object> l; dump(l);
- List<Integer> l; dump(l);

# ПРОБЛЕМА 2

- Метод

```
void draw(List<Shape> c) {  
    for (Iterator<Shape> i = c.iterator(); i.hasNext(); ) {  
        Shape s = i.next();  
        s.draw();  
    }  
}
```

- ВЫЗОВЫ

- List<Shape> l; draw(l);
- List<Circle> l; draw(l);

# РЕШЕНИЕ 2 – BOUNDED WILDCARD

- Метод

```
void draw(List<? extends Shape> c) {  
    for (Iterator<? extends Shape> i = c.iterator();  
         i.hasNext(); ) {  
        Shape s = i.next();  
        s.draw();  
    }  
}
```

- ВЫЗОВЫ

- List<Shape> l; draw(l);
- List<Circle> l; draw(l);

# ПРОБЛЕМА 3

- Метод

```
void addAll(Object[] a, Collection<?> c) {  
    for (int i = 0; i < a.length; i++) {  
        c.add(a[i]);  
    }  
}
```

- Примеры использования

- `addAll(new String[10], new ArrayList<String>());`
- `addAll(new Object[10], new ArrayList<Object>());`
- `addAll(new String[10], new ArrayList<Object>());`
- `addAll(new Object[10], new ArrayList<String>());`

# РЕШЕНИЕ 3 – GENERIC-МЕТОД

- Метод

```
<T> void addAll(T[] a, Collection<T> c) {  
    for (int i = 0; i < a.length; i++) {  
        c.add(a[i]);  
    }  
}
```

- Примеры использования

- `addAll(new String[10], new ArrayList<String>());`
- `addAll(new Object[10], new ArrayList<Object>());`
- `addAll(new String[10], new ArrayList<Object>());`
- `addAll(new Object[10], new ArrayList<String>());`



# ПРОБЛЕМА 4

- Метод

```
<T> void addAll(Collection<T> c, Collection<T> c2) {  
    for (Iterator<T> i = c.iterator(); i.hasNext(); ) {  
        T o = i.next();  
        c2.add(o);  
    }  
}
```

- Примеры использования

- `addAll(new AL<Integer>(), new AL<Integer>());`
- `addAll(new AL<Integer>(), new AL<Object>());`
- `addAll(new AL<Object>(), new AL<Integer>());`

# РЕШЕНИЕ 4 - BOUNDED TYPE ARGUMENT

- Метод

```
<T, S extends T> void addAll(Collection<S> c, Collection<T>  
    c2) {  
    for (Iterator<S> i = c.iterator(); i.hasNext(); ) {  
        S o = i.next();  
        c2.add(o);  
    }  
}
```

- Примеры использования

- `addAll(new AL<Integer>(), new AL<Integer>());`
- `addAll(new AL<Integer>(), new AL<Object>());`

# РЕШЕНИЕ 4' – BOUNDED WILDCARD

- Метод

```
<T> void addAll(Collection<? extends T> c, Collection<T> c2) {  
    for (Iterator<? extends T> i = c.iterator(); i.hasNext();  
        ) {  
        T o = i.next();  
        c2.add(o);  
    }  
}
```

- Примеры использования

- `addAll(new AL<Integer>(), new AL<Integer>());`
- `addAll(new AL<Integer>(), new AL<Object>());`

# ПРОБЛЕМА 5

- Метод

```
<T extends Comparable<T>>
```

```
    T max(Collection<T> c) {
```

```
    }
```

- Пример использования

- `List<Integer> il; Integer l = max(il);`

- `class Test implements Comparable<Object> {...}`

- `List<Test> tl; Test t = max(tl);`

# РЕШЕНИЕ 5 – UPPER BOUNDED WCARD

- Метод

```
<T extends Comparable<? super T>>  
    max(Collection<T> c) {  
        ...  
    }
```

- Пример использования

- `List<Integer> il; Integer l = max(il);`
- `class Test implements Comparable<Object> {...}`  
`List<Test> tl; Test t = max(tl);`

# WILDCARD CAPTURE (1)

```
void swap(List<?> list, int i, int j) {  
    // ?  
}
```

Не работает:

```
void swap(List<?> list, int i, int j) {  
    // Несоответствие типов  
    list.set(i, list.get(j));  
}
```

## WILDCARD CAPTURE (2)

```
void swap(List<?> list, int i, int j) {  
    swapImpl(list, i, j);  
}
```

```
<T> void swapImpl(List<T> list, int i, int j) {  
    T temp = list.get(i);  
    list.set(i, list.get(j));  
    list.set(j, temp);  
}
```