Generics

Java Features

Presented by Ramin Zare

بخش اول

Generic ها چه هستند؟

Generic ها چه هستند؟

Generics let you write true polymorphic code that works with any type!

به طور خلاصه،

با استفاده از Generic ها می توان هنگام تعریف کلاس ها، اینترفیس ها و متدها یک "نوع" را به عنوان پارامتر تعریف کرد.

اما اینکار چه فایده ای دارد؟

Generic ها چه هستند؟

```
public class ObjectWrapper {
    private Object ref;

    public ObjectWrapper(Object ref) {
        this.ref = ref;
    }

    public Object get() {
        return ref;
    }

    public void set(Object ref) {
        this.ref = ref;
    }
}
```

کلاس مقابل را در نظر بگیرید...
تنها وظیفه این کلاس ذخیره سازی یک reference به هرنوعی است.

در ادامه به بیان چند مشکل در مورد استفاده از Object برای ذخیره سازی یک reference برای انواع مختلف می پردازیم.

Generic ها چه هستند؟

تکه کد زیر کامپایل شده و بدون هیچ مشکلی اجرا می شود، در هر صورت با وجود اینکه می دانیم مقداری که قرار است در stringWrapper ذخیره شود از نوع String است اما باز هم کامپایلر ما را مجبور می کند که مقدار برگشتی را down cast کنیم!

```
ObjectWrapper stringWrapper = new ObjectWrapper("Hello");
stringWrapper.set("Another String");
String myString = (String) stringWrapper.get();
```

Generic ها چه هستند؟

تکه کد زیر نیز کامپایل می شود اما در زمان اجرا با ClassCastException روبرو خواهیم شد!

```
ObjectWrapper stringWrapper = new ObjectWrapper("Hello");
stringWrapper.set(new Integer(123));
String myString = (String) stringWrapper.get();
```

با استفاده از Generic ها برنامه نویسان می توانند برنامه های Type-Safe بنویسند!

با استفاده از Generic ها برنامه نویسان می توانند برنامه های Type-safe بنویسند!

```
public class Wrapper <T> {
    private T ref;

    public Wrapper(T ref) {
        this.ref = ref;
    }

    public T get() {
        return ref;
    }

    public void set(T ref) {
        this.ref = ref;
    }
}
```

کلاس ObjectWrapper را اما این بار به صورت Generic تعریف کرده ایم.

در اینجا T به معنای هر نوعی است، به عنوان مثال Person، Object، String و...

با استفاده از Generic ها برنامه نویسان می توانند برنامه های Type-safe بنویسند!

```
Wrapper<String> greetingWrapper = new Wrapper<String>("Hello");

//Ok to pass a String
greetingWrapper.set("Hi");

//No need to cast!
String myString = greetingWrapper.get();

//Compile-time error
//You can use greetingWrapper only to store a String.
greetingWrapper.set(new Integer(101));
```

چند نکته

- نام پارامترها براساس قوانین نامگذاری Identifier ها در جاوا هر مقداری می تواند باشد اما به صورت قراردادی از T برای مشخص کردن اینکه پارامتر یک type است و از E، K، N، V به ترتیب برای مشخص کردن اینکه پارامتر یک value ، number ، key و یا یک element است استفاده می شود.

- پارامترهای چند گانه را می توان با کاما از همدیگر جدا کرد.

```
public class MyClass<T, U, V, W> {
}
```

چند نکته

- Generic ها در زمان کامپایل پیاده سازی می شوند و JVM در زمان اجرا هیچ اطلاعی از generic type ها ندارد. به این فرآیند **erasure** یا **پاک شدگی** می گوییم.

با توجه به این فرآیند تمام اطلاعات مشخص شده برای genetic type ها در زمان کامپایل حذف می شوند. بنابراین مثلا `Wrapper<Long> a;` دقیقا معادل `Wrapper a;` خواهد بود!

به تکه کد بعدی توجه کنید!

چند نکته

Runtime Class Type of Generic Objects

```
package com.javaland.generics;
public class GenericsRuntimeClassTest {
    public static void main(String[] args) {
        Wrapper<String> a = new Wrapper<String>("Hello");
        Wrapper<Integer> b = new Wrapper<Integer>(new Integer(123));
        Class aClass = a.getClass();
        Class bClass = b.getClass();
        System.out.println("Class for a: " + aClass.getName());
        System.out.println("Class for b: " + bClass.getName());
        System.out.println("aClass == bClass: " + (aClass == bClass));
    }
}
```

```
Class for a: com.javaland.generics.Wrapper
Class for b: com.javaland.generics.Wrapper
aClass == bClass: true
```

چند نکته

- رابطه نرمالی که بین Supertype و Subtype در حالت عادی وجود دارد، در Generic ها برقرار نیست. باوجود اینکه String خود یک Object است اما Wrapper<String> یک Wrapper<Object> محسوب نمی شود! (assignment compatible نیست)

```
Wrapper<String> stringWrapper = new Wrapper<String>("Hello");
stringWrapper.set("a string");

Wrapper<Object> objectWrapper = new Wrapper<Object>(new Object());
objectWrapper.set(new Object());

// Use a String object with objectWrapper
objectWrapper.set("a string"); // OK

//Compile-time error
objectWrapper = stringWrapper;
```

چند نکته

- پیاده سازی Generic ها در جاوا backward compatible است. به عبارت دیگر اگر یک کلاس non-generic موجود به صورت یک کلاس generic بازنویسی شود، کدهایی که قبلا با آن کلاس کار می کرده اند بدون هیچ مشکلی باز هم کار خواهند کرد!

نسخه non-generic از یک نوع Generic را یک raw type می نامیم.

چند نکته

- اگر type parameter یک کلاس generic در هنگام تولید یک نمونه از آن کلاس مشخص نگردد در این صورت نوع آن یک raw type خواهد بود و کامپایلر unchecked warrning تولید خواهد کرد.

```
// Use the Wrapper<T> generic type as a raw type Wrapper
Wrapper rawType = new Wrapper("Hello"); // An unchecked warning
```

```
$ javac *.java -Xlint:unchecked
MainClass.java:117: warning: [unchecked] unchecked call to Wrapper(T) as a member of
the raw type Wrapper
        Wrapper rawType = new Wrapper("Hello"); // An unchecked warning
                          ^
  where T is a type-variable:
    T extends Object declared in class Wrapper
1 warning
```

چند نکته

- برای کلاس های generic و یا حتی non-generic می توان method ها و constructor های Generic تعریف کرد. برای این منظور type parameter های مورد نظر را قبل از نوع بازگشتی متد در <> قرار می دهیم.

```
public class Test<T> {
    public <U extends T> Test(U param) {
        // Do something
    }

    public <V> void m1(Wrapper<V> a, Wrapper<V> b, T c) {
        // Do something
    }
}
```

متد m1 کاربر استفاده کننده از آن را ملزم می کند دو پارامتر اول را از یک نوع و پارامتر سوم را از نوع T که همان نوع مشخص شده در زمان class instantiation است در نظر بگیرد.

چند نکته

متد m1 را به دو روش می توان فراخوانی کرد:

```
Test<String> t = new Test<String>("some text");

Wrapper<Integer> iw1 = new Wrapper<Integer>(new Integer(301));
Wrapper<Integer> iw2 = new Wrapper<Integer>(new Integer(555));

// Specify that Integer is the actual type for the type parameter for m1()
t.<Integer>m1(iw1, iw2, "hello");

// Let the compiler figure out the actual type parameters
// using types for iw1 and iw2
t.m1(iw1, iw2, "hello"); // OK
```

چند نکته

- type parameter های تعریف شده در کلاس generic در متدهای static آن کلاس قابل دسترس نیستند. بنابراین در صورت نیاز برای آن متد باید type parameter ها به صورت جدا تعریف شوند.

```
public class Test<T> {
    public static <V> void foo(V param) {
        // Do something
    }
}
```

بخش دوم

- استفاده از wildcard ها به عنوان actual type parameter ها.
- تعریف کران بالا و پایین برای type parameter ها.

استفاده از wildcard ها به عنوان actual type parameter ها

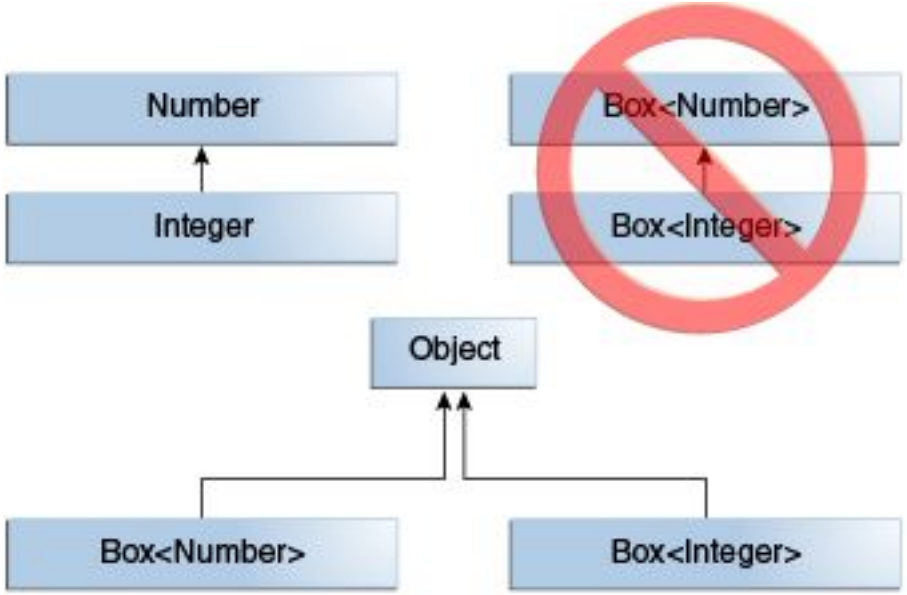
کلاس زیر را در نظر بگیرید، این کلاس دارای متدی است که یک شی از کلاس Wrapper<T> می گیرد.

```
public class WrapperUtil {
    public static void printDetails(Wrapper<Object> wrapper){
        // More code goes here
    }
}
```

اما از آنجاییکه Wrapper<String> و Wrapper<Object> را نمی توان به همدیگر assign کرد، تکه کد زیر خطای کامپایل خواهد داشت.

```
Wrapper<String> stringWrapper = new Wrapper<String>("Hello");
WrapperUtil.printDetails(stringWrapper); // A compile-time error
```

استفاده از wildcard ها به عنوان actual type parameter ها



استفاده از wildcard ها به عنوان actual type parameter ها

راه حل:

هنگامیکه در مورد parameter type یک Generic هیچ اطلاعی وجود نداشته باشد، می توان در هنگام تعریف آن، parameter type آن را با ? مشخص کرد. به <?> wildcard گفته می شود.

```
public class WrapperUtil {
    public static void printDetails(Wrapper<?> wrapper) {
        // More code goes here
}
```

استفاده از wildcard ها به عنوان actual type parameter ها

چند نکته:

```
// You can assign a Wrapper<String> to Wrapper<?> type
Wrapper<?> unknownWrapper = new Wrapper<String>("Hello"); // OK


// unknownWrapper does not know its type
Wrapper<?> unknownWrapper = new Wrapper<?>(""); // compile-time error


// unknownWrapper does not know its type
String str = unknownWrapper.get(); // compile-time error
```

استفاده از wildcard ها به عنوان actual type parameter ها

چند نکته :

```
// All reference types in Java are subtypes of the Object type
Object obj = unknownWrapper.get(); // OK


// Compiler can not make sure what the unknown type for unknownWrapper is
unknownWrapper.set(new Integer()); // compile-time error
unknownWrapper.set("Hello"); // compile-time error
unknownWrapper.set(new Object()); // compile-time error


// A null is assignment-compatible to any reference type
unknownWrapper.set(null); // OK
```

تعریف کران بالا و پایین برای type parameter ها

1- فرض کنید متدی مورد نیاز باشد که دو مقدار عددی که در شی Wrapper قرار دارند را گرفته و یک عملیات ریاضی مثلا جمع روی آن ها انجام دهد. یک روش اینکار به صورت زیر است:

```
public static double sum(Wrapper<?> n1, Wrapper<?> n2) {
    //Code goes here
}
```

همچنین فرض کنید این متد به صورت زیر فراخوانی شود!

```
// Try adding an Integer and a String
sum(new Wrapper<Integer>(new Integer(125)), new Wrapper<String>("Hello"));
```

باوجود اینکه کدهای فوق به درستی کامپایل می شوند. اما در زمان اجرا exception رخ می دهد.

تعریف کران بالا و پایین برای type parameter ها

2- فرض کنید متدی مورد نیاز باشد که دو شی Wrapper را گرفته و شی موجود در یکی را در دیگری کپی کند. یک روش اینکار به صورت زیر است:

```
public static <T> void copy(Wrapper<T> source, Wrapper<T> dest) {
        T value = source.get();
        dest.set(value);
}
```

همچنین فرض کنید این متد به صورت زیر فراخوانی شود!

```
copy(new Wrapper<String>("Hello"), new Wrapper<Object>(new Object()));
```

باتوجه به اینکه دو پارامتر ورودی هم نوع نیستند، کد بالا کامپایل نخواهد شد.

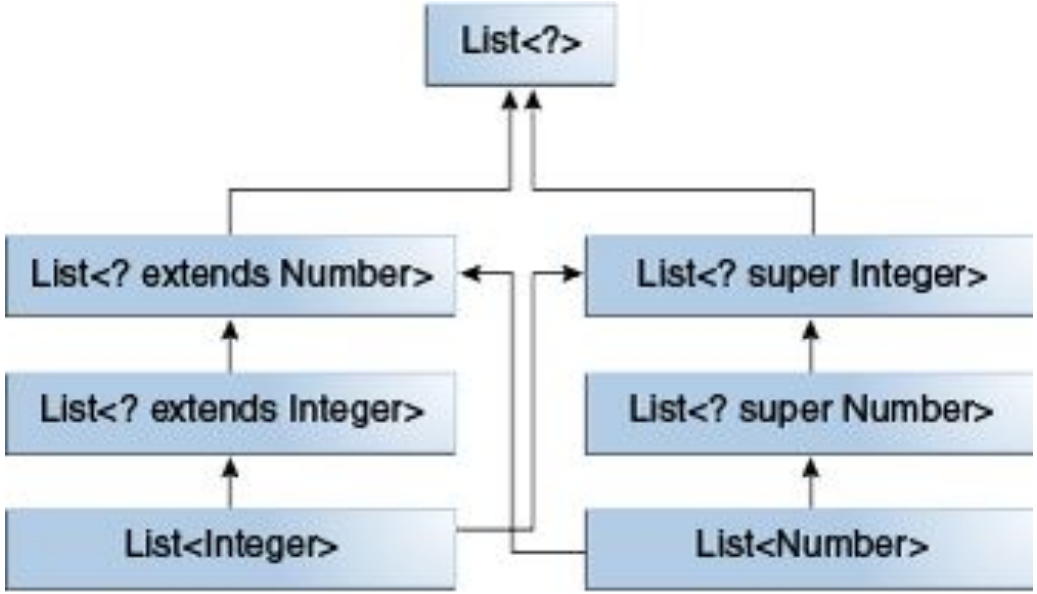
تعریف کران بالا و پایین برای type parameter ها

با تعریف یک کران بالا/پایین برای مقادیر قابل قبول، می توان محدودیت بیشتری روی کدها در زمان کامپایل اعمال کرد و از بروز exception های زمان اجرا جلوگیری کرد.

کران بالا به صورت `<? extends T>` تعریف شده و به معنای "نوع T و یا همه sub-type های T است".

کران پایین به صورت `<? super T>` تعریف شده و به معنای "نوع T و یا همه super-type های T است".

تعریف کران بالا و پایین برای type parameter ها



تعریف کران بالا و پایین برای type parameter ها

متد sum را با مشخص کردن کران بالا برای پامترهای آن، به صورت زیر بازنویسی می کنیم:

```
public static double
            sum(Wrapper<? extends Number> n1, Wrapper<? extends Number> n2) {

    Number num1 = n1.get();
    Number num2 = n2.get();
    double sum = num1.doubleValue() + num2.doubleValue();
    return sum;

}
```

تعریف کران بالا و پایین برای type parameter ها

متد copy را با مشخص کردن کران پایین برای پامتر دوم آن، به صورت زیر بازنویسی می کنیم:

```
public static <T> void copy(Wrapper<T> source, Wrapper<? super T> dest){
    T value = source.get();
    dest.set(value);
}
```

بخش سوم

- چگونه کامپایلر actual type parameter ها را از روی نحوه تعریف generic type ها استنتاج می کند؟
- محدودیت های Generic ها چیست؟
- چگونه استفاده نادرست از Generic ها باعث آلودگی heap می گردد؟

چگونه کامپایلر actual type parameter ها را از روی نحوه تعریف generic type ها استنتاج می کند؟

تکه کد زیر را در نظر بگیرید:

```
T1<T2> var = new T3<>(constructor-arguments);
```

1- در ابتدا کامپایلر تلاش می کند مقدار type parameter را بر اساس static type آرگومان ورودی constructor تشخیص دهد. توجه کنید ممکن است constructor آرگومان ورودی نداشته باشد.

چگونه کامپایلر actual type parameter ها را از روی نحوه تعریف generic type ها استنتاج می کند؟

تکه کد زیر را در نظر بگیرید:

```
T1<T2> var = new T3<>(constructor-arguments);
```

2- در این مرحله کامپایلر تلاش می کند از روی عبارت سمت چپ عملگر تساوی مقدار type parameter را تشخیص دهد (در تکه کد بالا T2 تشخیص داده می شود).

اگر عبارت تولید شی، به صورت یک assignment statement نباشد به مرحله بعد می رود.

چگونه کامپایلر actual type parameter ها را از روی نحوه تعریف generic type ها استنتاج می کند؟

```
List<String> stringList = Arrays.asList("A", "B");
List<Integer> integerList = Arrays.asList(9, 19, 1969);


// Inferred type is String
List<String> list3 = new ArrayList<>(stringList);


// compile-time error
List<String> list4 = new ArrayList<>(integerList);


// Inferred type is String
List<String> list5 = new ArrayList<>();
```

چگونه کامپایلر actual type parameter ها را از روی نحوه تعریف generic type ها استنتاج می کند؟

3- اگر عبارت تولید شی به صورت فراخوانی یک متد باشد، کامپایلر تلاش می کند مقدار type parameter را بر اساس پارامترهای ورودی متد فراخوانی شده تشخیص دهد.

4- در صورتیکه همه مراحل قبل با شکست مواجه شوند، کامپایلر java.lang.Object را به عنوان مقدار type parameter در نظر می گیرد.

محدودیت های Generic ها چیست؟

1- کلاس های Exception را نمی توان به صورت Generic تعریف کرد.

با توجه به اینکه exception ها در زمان اجرا پرتاب می شوند و چون در طی کامپایل تمام mention های موجود به type parameter ها پاک می شوند پس نمی توان از type-safe بودن exception ها در زمان اجرا اطمینان حاصل کرد.

محدودیت های Generic ها چیست؟

2- Anonymous Class ها را نمی توان به صورت Generic تعریف کرد.

با توجه به اینکه Anonymous Class ها بدون نام هستند، نمی توان actual type parameter برای آن ها مشخص کرد. اما می توان در Anonymous Class متدهای generic داشت.

3- Enum ها را نمی توان به صورت Generic تعریف کرد.

محدودیت های Generic ها چیست؟

4- نمی توان آرایه ای از نوع generic type تعریف کرد.

یک آرایه در زمان ایجاد شدن باید از نوع خودش اطلاع داشته باشد تا در زمان ذخیره سازی عناصر در آن بتوان assigment compatible بودن عناصر با آرایه را بررسی کرد و چون در زمان کامپایل تمام generic type parameter ها به reference ها از کد حذف می شوند، این اطلاع در زمان run-time دیگر وجود نخواهد داشت.

```
Wrapper<String>[] a = new Wrapper<String>[10]; // compile-time error
```

نکته: تولید یک آرایه با استفاده از unbounded wildcard ها کاملا مجاز است.

```
Wrapper<?>[] b = new Wrapper<?>[10]; // Ok
```

چگونه استفاده نادرست از Generic ها باعث آلودگی heap می گردد؟

آلوده شدن فضای Heap وضعیتی است که یک متغیر از یک جنس parameterized type به یک شی که از همان جنس نباشد اشاره کند.

باتوجه به اینکه اطلاعات generic type ها در زمان کامپایل طی فرآیند **erasure** حذف می شوند احتمال آلوده شدن heap وجود دارد، در این حالت کامپایلر با تولید unchecked warrning از احتمال آلوده شدن heap در زمان اجرا خبر می دهد!

چگونه استفاده نادرست از Generic ها باعث آلودگی heap می گردد؟

به تکه کد زیر توجه کنید:

```
1.  Wrapper nWrapper = new Wrapper<Integer>(101);
2.  Wrapper<String> strWrapper = nWrapper;
3.  String str = strWrapper.get();
```

خط 1 بدون خطا کامپایل می شود (عبارت سمت چپ تساوی به صورت raw type تعریف شده است)

خط 2 با ایجاد Unchecked warning در زمان کامپایل، از آلوده شدن heap در زمان اجرا خبر می دهد.

خط 3 در زمان اجرا ClassCastException تولید می کند.

چگونه استفاده نادرست از Generic ها باعث آلودگی heap می گردد؟

به تکه کد زیر توجه کنید:

```
1.  Wrapper<? extends Number> nWrapper = new Wrapper<Long>(1L);
2.  Wrapper<Short> sWrapper = (Wrapper<Short>) nWrapper;
3.  short str = sWrapper.get();
```

خط 1 بدون خطا کامپایل می شود.

خط 2 به دلیل انجام عملیات Unchecked Cast در این خط کامپایلر با تولید Unchecked warning،

از آلوده شدن heap در زمان اجرا خبر می دهد.

خط 3 در زمان اجرا ClassCastException تولید می کند.

چگونه استفاده نادرست از Generic ها باعث آلودگی heap می گردد؟

به متد زیر توجه کنید:

```
public static void process(Wrapper<Long>... nums) {
    Object[] obj = nums;                  // Heap pollution
    obj[0] = new Wrapper<>("Hello"); // Array corruption
    Long lv = nums[0].get();              // ClassCastException
}
```

با توجه به اینکه جاوا پارامترهای vararg را با تبدیل آنها به آرایه پیاده سازی کرده است در صورت استفاده از نوع generic برای این گونه پارامترها نمی توان type-safety را تضمین کرد و ممکن است باعث آلودگی Heap و ایجاد Exception در زمان اجرا گردد.

Thanks!

Presented by :

Ramin Zare

Thanks to :

Ali Khaleghzadegan

Mohammad Taghizadeh

Mahdi Abedi