



پر نامه سازی پیش رفته

انواع داده و متدهای عام

مهدی مصطفی زاده

سرفصل مطالب

- رده‌های عام (Generic Classes)
- محدودسازی پارامتر نوع
- ارث بری انواع عام و غیر عام از یکدیگر
- متدهای عام (Generic Methods)
- فرایند محو (Erasure)



طرح مساله

- گاهی اوقات نیاز به پیاده‌سازی رده‌هایی داریم که منطق (ساختار و رفتار) آن‌ها کاملاً یکسان است.
- فقط پر سر یک یا چند نوع داده‌ای با هم تفاوت دارند.
- مثال: فهرست (لیست) داشتچویان، فهرست دروس، فهرست کتب، ...
- راهکار اول: پیاده‌سازی هر رده به شکل معجزاً مثلاً BooksList، CoursesList، StudentsList، ...
- مشکل: افرایش پیچیدگی برنامه (از منظر تعداد رده‌ها)، کد تکراری و در نتیجه، انتشار تغییرات (کاهش قابلیت نگهداری)



طرح مساله (ادامه)

- راهکار دوم: پیاده‌سازی همه‌ی رده‌ها در قالب یک رده و چایگزین کردن همه‌ی انواع متغیر با رده‌ی Object. مثلاً: List<Object>.
- مشکل: احتمال اشتباه پر نامه‌نویس در ایجاد شئ‌ای با حالت نامعتبر (تدریکی از اشیاء از انواع داده‌ی مختلف)
- مثال: مطلوب آن است که کامپایلر مانع آن شود که پر نامه‌نویس شئ‌ای از نوع کتاب را درون لیست دروس درج کند.
- راهکار: استفاده از امکان انواع داده‌ی عام (Generic Data Types) در چاوا (C++) = Templates در زبان



نوع داده‌ی عام

- رده‌ای که در تعریف خود دارای یک یا چند پارامتر نوع (Type Parameter) است.
- به آن نوع داده‌ی پارامتردار (Parameterized Data Type) نیز می‌گویند.
- پردازه‌نویس، هنگام استفاده از رده، هر یک از پارامترها را به یک نوع داده‌ی ارجاعی (Referenced Data Type) مُقید می‌کند.
- انواع داده‌ی اولیه (Primitive Data Types) نمی‌توانند به عنوان پارامتر ارسال شوند.



رده‌های لفاف انواع داده‌ی اولیه

- در صورت نیاز، به جای انواع داده‌ی اولیه، استفاده از رده‌های لفاف آنها (Primitive Wrapper Classes) ممکن است.

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

پسته‌پندی خودکار (Autoboxing)

Integer num = 5

پازگشایی (Unboxing)

int num = new Integer(5)

خطای کامپایل!
Long num = 5



تعریف و استفاده از نوع داده‌ی عام

```
5 public class List<E> {
6     E[] items;
7
8     int numberOfItems = 0;
9
10    public List(int capacity) {
11        items = (E[]) new Object[capacity];
12    }
13
14    public void add(E item) {
15        items[numberOfItems++] = item;
16    }
17
18    public E get(int index) {
19        return items[index];
20    }
21 }
```

```
public class List<E> {
    E[] items;

    int numberOfItems = 0;

    public List(int capacity) {
        items = new E[capacity];
    }
    ...
}

List<Integer> integers = new List<>(5);
integers.add(7);
System.out.println(integers.get(0));
```



تعریف و استفاده از نوع داده‌ی عام (چند پارامتر)

```
public class Entry<K, V> {  
    private K key;  
    private V value;  
  
    public Entry(K key, V value) {  
        this.key = key;  
        this.value = value;  
    }  
  
    public K getKey() {  
        return key;  
    }  
  
    public V getValue() {  
        return value;  
    }  
}
```

```
Entry<Integer, String> student =  
    new Entry<Integer, String>(10, "Ai Ahmadi");  
System.out.println(student.getValue());
```



محدودسازی پارامتر نوع

```
5 public class List<E extends Comparable<E>> {      public class List<E extends Number> {
6
7     public List(int capacity) {
8         items = (E[]) new Comparable[capacity];
9     }
10
11    public void sort() {
12        for(int i = 1; i < numberOfItems; i++) {
13            E temp = items[i];
14            int j;
15            for(j = i - 1; j >= 0 && items[j].compareTo(temp) > 0; j--) {
16                items[j + 1] = items[j];
17            }
18            items[j + 1] = temp;
19        }
20    }
}
```



ارث بری انواع عام و غیر عام از یکدیگر

```
class A {  
}  
  
class B<E> extends A {  
}  
  
class A<T> {  
}  
  
class B extends A<Person> {  
}  
  
class A<T> {  
}  
  
class B<E extends Number, T> extends A<E> {  
}
```

- پیش از این دیده ایم که چطور یک نوع غیر عام می تواند فرزند یک نوع غیر عام دیگر باشد.
- یک نوع عام می تواند فرزند یک نوع غیر عام باشد.
- یک نوع غیر عام می تواند فرزند یک نوع عام باشد؛ به شرطی که مقدار پارامتر (های) نوع را مشخص کرده باشد.
- یک نوع عام می تواند فرزند یک نوع عام دیگر باشد (فرزند عام می تواند پارامتر نوع پدرش را محدود تر کند).



استفاده از نوع عام به شکل خام

- می‌توان بدون تعیین پارامتر نوع، از انواع عام استفاده کرد.
- به آن نوع خام (Raw Type) می‌گویند.
- در صورت استفاده از نوع خام، کامپایلر کمترین محدودیت Comparable، Number، Object یا ... را پر روی آن اعمال می‌کند.



متدهای عام (Generic Methods)

- گاهی اوقات، در یک رده نیاز به تعریف متدهایی داریم که منطق آن‌ها کاملاً یکسان است، اما تفاوت آن‌ها بر سر تفاوت نوع پارامترها و نوع خروجی است.
- متدهای عام، امکان تعریف یک‌چای همه‌ی این متدها را فراهم می‌آورد. (لزومی ندارد که رده نیز عام باشد)
- در این متدها، معمولاً نوع خروجی پرساسن نوع پارامتر ورودی تعیین می‌شود.

```
public class RelationUtil {  
    public <E extends Comparable<E>> E getLargest(E e1, E e2) {  
        return e1.compareTo(e2) > 0 ? e1 : e2;  
    }  
}
```



فرایند محو (Erasure)

- انقیاد پارامتر نوع به یک نوع مشخص به منظور اعمال کنترل توسط کامپایلر، فقط در زمان کامپایل موضوعیت دارد.
- در زمان اجرا، هیچ ردی از پارامتر نوع نیست.
- در زمان اجرا، تفاوتی بین پایت کدهای `List<Courses>` و `List<Student>` و په جای پارامتر نوع، کلی تدین حالت (پا توجه به محدوده Comparable، Object یا ...) قرار می گیرد.
- در زمان اجرا مشخص نیست که یک شئ پا چه پارامتری ایجاد شده است.
- په نادیده گرفته شدن پارامتر نوع در زمان اجرا توسط چاوا، فرایند محو (Erasure) می گویند.



محدودیت‌های ناشی از فرایند محو

```
public <E extends Number> void f(E e1, E e2) {  
    E e = new E();  
}                                              ✘ Cannot instantiate the type E  
  
public <E extends Number> void f(E e1, E e2) {  
    E[] e = new E[10];  
}                                              ✘ Cannot create a generic array of E  
  
public class RelationUtil {  
    public <E extends Number> void f(E e1, E e2) {  
    }  
  
    public void f(Number e1, Number e2) {  
    }                                              ✘ Erasure of method f(Number, Number) is the same as another method in type RelationUtil  
}
```

- نمونه‌سازی (new) از روی پارامتر نوع

- تعریف متدهای همنام
(Method Overloading)



محدودیت‌های ناشی از فرایند محو (ادامه)

```
public <E extends Number> void f(E e1, E e2) {  
    if(e1 instanceof E) {  
    }  
}
```

Cannot perform instanceof check against type parameter E. Use its erasure Number instead since further generic type information will be erased at runtime
Press 'F2' for f

عملگر instanceof

روی پارامتر نوع

```
public class List<E extends Comparable<E>> {  
    static E e;  
    ...  
}
```

Cannot make a static reference to the non-static type E

تعریف متغیر ایستا با پارامتر نوع



محدودیت‌های ناشی از فرایند محو (ادامه - ۲)

```
List<Person> list = new List<>(10);  
  
if(list instanceof List) { // is valid  
}  
  
if(list instanceof List<Person>) {  
}  
  ❌ Cannot perform instanceof check against parameterized type List<Person>, Use the form List<?> instead  
  since further generic type information will be erased at runtime  
  
List[] list = new List[10];  
List<Person>[] list2 = new List<Person>[10];  
  ❌ Cannot create a generic array of List<Person>
```

عملگر روی نوع عام

نمونه‌سازی آرایه از روی نوع عام



محدودیت‌های ناشی از فرایند محو (ادامه - ۳)

- یک نوع عام نمی‌تواند (په طور مستقیم یا غیرمستقیم) فرزنده Throwable باشد. په عبارت دیگر، نوع عام نمی‌تواند به عنوان استثناء، مورد استفاده قرار گیرد.

```
public class List<E> extends Exception {  
    ☹ The generic class List<E> may not subclass java.lang.Throwable
```

- پارامتر نوع (و نه خود نوع عام) می‌تواند به عنوان نوع استثناء، مورد استفاده قرار گیرد. پدیوهی است که بازهم نمی‌تواند new شود.

```
public class List<E extends Throwable> {  
    public void sort() throws E {
```

