انجمن جاواکاپ تقدیم می‌کند

دوره برنامه‌نویسی جاوا

انواع داده عام
Generics

صادق علی اکبری

حقوق مؤلف

- کلیه حقوق این اثر متعلق به انجمن جاواکاپ است
- بازنشر یا تدریس آن‌چه توسط جاواکاپ و به صورت عمومی منتشر شده است، با ذکر مرجع (جاواکاپ) بلامانع است
- اگر این اثر توسط جاواکاپ به صورت عمومی منتشر نشده است و به صورت اختصاصی در اختیار شما یا شرکت شما قرار گرفته، بازنشر آن مجاز نیست
- تغییر محتوای این اثر بدون اطلاع و تأیید انجمن جاواکاپ مجاز نیست

سرفصل مطالب

- کلاس‌های عام (Generic Classes)
- متدهای عام (Generic Methods)
- انواع عام و وراثت
- فرایند مَحو (Erasure)

چه نیازی به انواع عام است؟

انواع داده عام (Generic)

- گاهی منطق پیاده‌سازی یک کلاس، برای انواع داده مختلف یکسان است
- مثلاً منطق متد add در کلاس ArrayList به ازای لیستی از رشته یا عدد متفاوت نیست
- راه اول: به ازای هر نوع داده، یک کلاس ArrayList بسازیم
  - مثلاً: StringArrayList و IntegerArrayList و StudentArrayList
  - این کلاس‌ها مشابه (کپی) یکدیگرند — Code redundancy
- راه دوم: یک کلاس ArrayList تعریف کنیم و درهنگام استفاده نوع آن را محدود کنیم

```
ArrayList<String> list1 = new ArrayList<String>();
list1.add("Ali");
ArrayList<Integer> list2 = new ArrayList<Integer>();
list2.add(new Integer(2));
```

- کلاس ArrayList یک **نوع داده عام (Generic)** است
- بحث این جلسه: چگونه کلاس‌های عام (مثل ArrayList) تعریف کنیم؟

مسأله چیست؟

- فرض کنید می‌خواهیم کلاس «ظرفی از اشیاء» طراحی کنیم
  - مثلاً یک لیست یا مجموعه یا پشته یا صف
- این کلاس متدهایی مثل add و delete خواهد داشت. مثال:

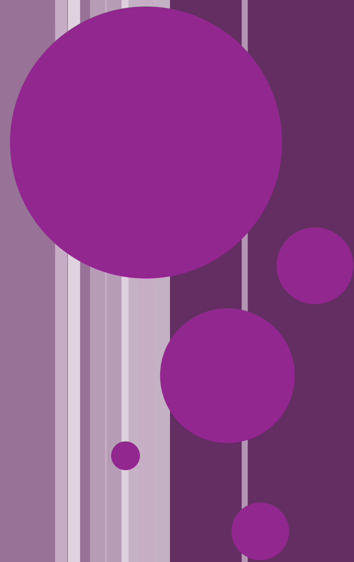
```
MyList q = new MyList();
q.add("Ali");
q.add("Taghi");
```

- اما معمولاً هر ظرف شامل اشیائی از یک نوع یکسان است
  - مثلاً لیستی از «رشته»ها یا مجموعه‌ای از «عدد»ها یا یک صف از «دانشجو»ها
- چنین نیازی را چگونه پیاده می‌کنید؟ مثلاً متد add چگونه اعلان شود؟
- مثلاً این تعریف مناسب است؟

```
void add(Object obj) {...}
```

مسأله چیست؟ (ادامه)

- فرض کنید می‌خواهیم کلاس MyList یا ArrayList را تعریف کنیم
- متد add را چگونه تعریف کنیم؟
- اگر این متد این‌گونه باشد: `void add(String obj) {...}`
  - این کلاس فقط برای رشته‌ها کار خواهد کرد (لیستی از رشته‌ها)
- اگر این متد این‌گونه باشد: `void add(Object obj) {...}`
  - نوع اشیاء این کلاس محدودیتی ندارد (لیستی از هر نوع شیء)
  - ممکن است در یک لیست، همزمان اشیائی از نوع رشته، عدد یا دانشجو داشته باشیم
- معمولاً علاقمندیم یک ظرف (مثلاً لیست) ، اشیائی از یک نوع داشته باشد
- مثل ظرف‌های جاوا (ArrayList، HashSet و ...) : این ظرف‌ها چگونه تعریف شده‌اند؟



تعریف انواع داده عام

نحوه تعریف کلاس عام

```java
public class Stack<E> {
 private E[] elements;
 private int top;
 public void push(E pushValue) {
  if (top == elements.length - 1) throw new FullStackException();
  elements[++top] = pushValue;
 }
 public E pop() {
  if (top == -1) throw new EmptyStackException();
  return elements[top--];
 }
 public Stack(int maxsize) {
  top = -1;
  elements = (E[]) new Object[maxsize];
 }
}
```

پارامتر نوع
(Type Parameter)

```java
Stack<String> st1;
st1 = new Stack<String>(10);
st1.push("A");
st1.push("B");
String p1 = st1.pop();
```

```java
Stack<Integer> st2;
st2 = new Stack<Integer>(10);
st2.push(new Integer(1));
st2.push(new Integer(2));
Integer p2 = st2.pop();
st2.push("A");
```

مثال: کلاس Pair

```java
public class Pair<T1, T2> {
    private T1 first;
    private T2 second;
    public T1 getFirst() {return first;}
    public void setFirst(T1 first) {this.first = first;}
    public T2 getSecond() {return second;}
    public void setSecond(T2 second) {this.second = second;}
    public Pair(T1 first, T2 second) {
        this.first = first;
        this.second = second;
    }
}
```

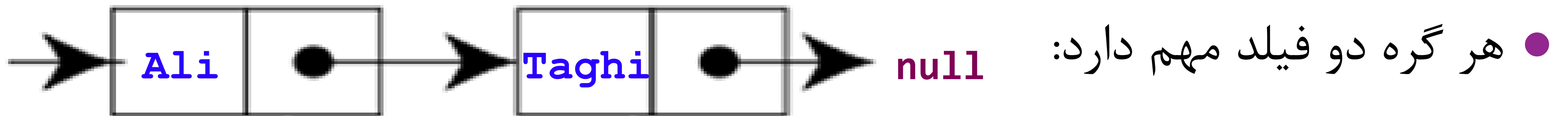
- فرض کنید می‌خواهیم کلاس Pair تعریف کنیم
  - هر شیء از این کلاس یک جفت شیء (زوج مرتب) در درون خود نگه می‌دارد
  - می‌خواهیم در زمان ایجاد شیء (و نه در زمان تعریف کلاس) نوع این دو شیء را تعیین کنیم

```java
Pair<String, Double> p1;
p1 = new Pair<String, Double>("Ali", 19.0);
String name = p1.getFirst();
Double avg = p1.getSecond();
```

```java
Pair<String, String> p2;
p2 = new Pair<String, String>("Ali", "Alavi");
String fname = p2.getFirst();
String lname = p2.getSecond();
```

مثال

- برای هر گره از یک لیست پیوندی، می‌خواهیم یک کلاس با نام Node تعریف شود
- هر گره دو فیلد مهم دارد:



۱- مقدار (از هر نوعی می‌تواند باشد)

۲- ارجاع به گره بعدی (ارجاعی به یک Node)

```java
class Node<E> {
    E item;
    Node<E> next;
    Node(E element, Node<E> next) {
        this.item = element;
        this.next = next;
    }
}
```

```java
Node<String> last = new Node("Taghi", null);
Node<String> first = new Node("Ali", last);
```

مرور چند واسط و کلاس عام جاوا

مثال

```java
class ArrayList<E> implements List<E>
//extends... implements...
{
   public int size() {...}
   public E get(int index) {...}
   public E set(int index, E element) {...}
   public boolean add(E e) {...}
    ...
}
```

```java
interface List<E> extends Collection<E> {
    int size();
    boolean isEmpty();
    boolean add(E e);
    boolean equals(Object o);
    E get(int index);
    E set(int index, E element);
    void add(int index, E element);
    E remove(int index);
    List<E> subList(int fromIndex, int toIndex);
}
```

مانند یک کلاس، یک واسط یا یک کلاس مجرد هم می‌تواند عام (generic) باشد

مثال

```java
interface Map<K,V> {
    int size();
    V get(Object key);
    V put(K key, V value);
    V remove(Object key);
    Set<K> keySet();
    Collection<V> values();
}
```

امکان استفاده از چند «پارامتر نوع»
در یک کلاس عام (generic type)

```java
public class HashMap<K,V> implements Map<K,V>
//extends... implements...
{
 ...
}
```

چند نکته درباره انواع داده عام

نکته

- هنگام تعریف متغیر از یک نوع عام، می‌توانیم نوع عام را با یک نوع مشخص تعیین کنیم
- مثلاً واسط List را در نظر بگیرید:

```
interface List<E>
```

- هنگام ایجاد متغیر از جنس List ، می‌توانیم E را با نوع موردنظر جایگزین کنیم:

```
List<String> strs;
List<Integer> ints;
List<Student> stus;
```

- اما به عنوان «پارامتر نوع»، نمی‌توانیم از انواع داده اولیه (مثل double) استفاده کنیم
  - جایگزین E در مثال فوق فقط یک «کلاس» می‌تواند باشد
- خطای کامپایل:

```
List<int> error1;
List<double> error2;
```

فایده انواع داده عام

- در زمان کامپایل، از اشتباه برنامه‌نویس جلوگیری می‌کند (پیش از اجرای برنامه)
- اگر برنامه‌نویس متغیر strs را به این شکل تعریف کند: `List<String> strs;`
  یعنی قرار است strs لیستی از رشته‌ها باشد
- با ذکر پارامتر نوع (رشته) و نظارت کامپایلر، برنامه نویس نمی‌تواند سهواً اشتباه کند
- مثلاً برنامه‌نویس نمی‌تواند `strs.add(new Integer(5))` را فراخوانی کند
- زیرا کامپایلر با یک syntax error جلوی آن را می‌گیرد
- به نوع عام (Generic Type) نوع پارامتردار (Parameterized Types) هم می‌گویند
  - در مثال فوق، List یک نوع عام یا پارامتردار است: String به عنوان پارامتر نوع
- نکته: داده عام از نسخه ۵ به جاوا اضافه شد. قبل از JDK1.5 اصلاً Generic نداشتیم

محدود کردن نوع عام

- هنگام استفاده از یک نوع عام، از هر کلاسی به عنوان پارامتر نوع می‌توانیم استفاده کنیم
- ولی گاهی نیازمندیم که پارامتر نوع را محدود به انواع خاصی کنیم
- مثال:

```
class NumbersQueue<T extends Number>{...}
```

```
inetrface SortedList<E extends Comparable>{...}
```

- در این‌جا، extends یعنی «پارامتر نوع» باید زیرکلاس یا زیرواسط نوع مشخص‌شده باشد
- مثال: فرض کنید:

```
class Person{}
```

```
NumbersQueue<Integer> n; ✔
NumbersQueue<Double> d; ✔
NumbersQueue<String> s; ✘
NumbersQueue<Person> p; ✘
```

```
SortedList<Long> l; ✔
SortedList<Float> f; ✔
SortedList<String> s; ✔
SortedList<Person> p; ✘
```

نوع خام (Raw Type)

- انواع داده عام را بدون تصریح «پارامتر نوع» هم می‌توان استفاده کرد
- در این صورت، کامپایلر حداقلِ محدودیت ممکن را برای این انواع اعمال می‌کند
- در این مثال، هر شیئی قابل افزودن به List است (محدودیتی نیست)

```
ArrayList list = new ArrayList();
list.add("A");
list.add(new Integer(5));
list.add(new Character('#'));
```

- در این مثال، فقط اشیائی از نوع Number (یا فرزندان Number) قابل استفاده در queue هستند

```
class NumbersQueue<T extends Number>{
 public void enqueue(T o){}
 public T dequeue(){...}
}
```

```
NumbersQueue queue;
queue = new NumbersQueue();
queue.enqueue(new Integer(1));
queue.enqueue(new Double(3.14));
queue.enqueue("Ali");
```

نکته

- از نسخه ۷ (java 1.7) به بعد، «استنتاج نوع» برای انواع عام ممکن شده است
- Type Inference
- به‌ویژه، ذکر نوع عام در هنگام نمونه‌سازی از انواع عام لازم نیست (نوع آن استنتاج می‌شود)
  - به این امکان، عملگر لوزی (diamond operator) می‌گویند
- مثال:

```
ArrayList<String> list = new ArrayList<String>();   قدیمی
ArrayList<String> list = new ArrayList<>();   جدید

Map<String, List<Student>> map =   قدیمی
  new HashMap<String, List<Student>>();
Map<String, List<Student>> map = new HashMap<>();   جدید
```

وراثت و انواع داده عام

یک نوع عام یا غیرعام می‌تواند از
یک نوع عام یا غیرعام ارث‌بری کند

۱- یک نوع غیرعام، فرزند نوع غیرعام دیگری باشد

```
class A{}
class B extends A{}
```

۲- یک نوع عام، فرزند یک نوع غیرعام باشد

```
class Box<T> extends B{}
```

۳- یک نوع غیرعام،
فرزند یک نوع عام باشد

```
class IntList implements List<Integer>{
  public boolean add(Integer e){…}
}
```

- در این صورت، زیرکلاس عام بودن را کنار می‌گذارد
- زیرکلاس تعیین می‌کند از چه نوع خاصی به جای پارامتر نوع اَبَرکلاس استفاده می‌کند

۴- یک نوع عام، فرزند یک نوع عام باشد

- در این صورت می‌تواند پارامتر نوع را محدودتر کند

```
interface NumberList<T extends Number> extends List<T>{}
```

متدهای عام

متد عام (Generic Method)

- دیدیم متدهای یک کلاس عام می‌توانند از نوع داده عام آن کلاس استفاده کنند
  - به عنوان پارامتر یا نوع داده برگشتی:

```
class ArrayList<E> {
  public E get(int index) {...}
  public boolean add(E e) {...}
}
```

- اما یک متد، خود نیز می‌تواند نوعی عام را به عنوان «پارامتر نوع» معرفی کند
  - نوعی غیر از آن چه در کلاس تعیین شده (مثلاً نوعی غیر از E در کلاس فوق)
  - حتی برای متدهایی که در کلاس‌های غیرعام قرار دارند
  - به این متدها، متد عام (Generic Method) گفته می‌شود

```
interface NotGeneric{
  public <E> E f(E p1);
}
```

```
class GenericClass<T>{
  public <E> void f(E p1, T p2){}
}
```

مثال

```java
class NotGeneric{
 public <T> T chooseRandom(T p1, T p2){
   if(new Random().nextFloat()>0.5)
    return p1;
   return p2;
 }
 public static <E extends Comparable> E max(E p1, E p2){
   return p1.compareTo(p2) > 0 ? p1 : p2;
 }
}
```

```java
String s = new NotGeneric().chooseRandom("A", "B");
Integer num = new NotGeneric().chooseRandom(1, 2);
num = NotGeneric.max(1, 2);
s = NotGeneric.max("A", "B");
```

کوییز

کوییز

- فرض کنید از شما خواسته شده واسط Set و کلاس HashSet را تعریف کنید
  - می‌دانیم Set و HashSet از انواع عام، و دارای متد add هستند
  - فرض کنید متد دیگری در این انواع وجود ندارد
- تعریف واسط Set و کلاس HashSet را بنویسید (هر کدام در سه خط)
  (تعریف بدنه add در HashSet لازم نیست)
- پاسخ:

```
interface Set<E> extends Collection<E> {
 void add(E e);
}
```

```
class HashSet<T> implements Set<T> {
 public void add(T e) {…}
}
```

(البته این پاسخ با تعریف واقعی Set و HashSet تفاوت‌هایی دارد)

کوییز

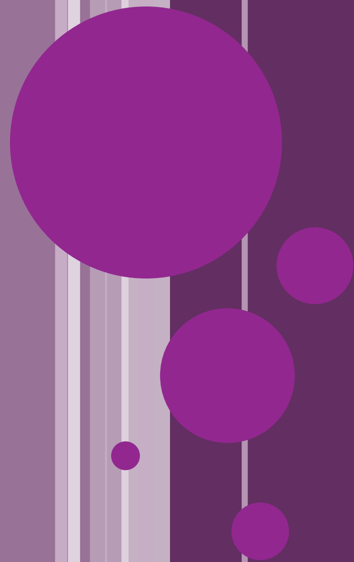
- می‌دانیم واسط Map به شکل زیر تعریف شده است:

```
interface Map<K,V> {…}
```

- امضای متدهای keySet و values در این واسط چگونه است؟
  - یادآوری: keySet کلیدها و values مقادیر موجود در map را برمی‌گردانند

- پاسخ:

```
Set<K> keySet();
Collection<V> values();
```



تمرین عملی

تمرین

- کلاس Box<T> را پیاده‌سازی کنید
  - استفاده از پارامتر نوع به عنوان نوع فیلد، پارامتر و مقدار برگشتی
  - تعریف سازنده
- متدهای عام (با پارامتر نوع T و غیر T) برای آن ایجاد کنید
- از این کلاس استفاده کنید
  - با انواع داده مختلف
  - به شکل خام
  - تأکید بر عدم امکان استفاده از انواع داده اولیه

مکانیزم مَحو (Erasure)

فرایند مَحو (Erasure)

- کنترل انواع داده عام (Generics) فقط مربوط به زمان کامپایل است
- در زمان اجرا، اطلاع و اثری از «پارامتر نوع» (Type Parameter) نیست

- اگر از یک «پارامتر نوع» برای ایجاد یک شیء استفاده کنیم،
  این «پارامتر نوع» فقط توسط کامپایلر چک می‌شود
- در زمان اجرا اثری از این «پارامتر نوع» نیست
- مثلاً در زمان اجرا معلوم نیست که یک شیء از نوع Stack به شکل Stack<String> ایجاد (new) شده یا به شکل Stack<Integer>
- در bytecode (فایل کامپایل‌شده یا class.) اطلاعاتی درباره «پارامتر نوع» یک شیء نیست
- در واقع همه انواع عام، به صورت «نوع خام» (raw type) خود ترجمه می‌شوند
- به این رفتار جاوا در قبال انواع عام، **فرایند مَحو** (**Erasure**) گفته می‌شود

فرایند مَحو (ادامه)

- وقتی کامپایلر، کدی شامل داده عام را ترجمه می‌کند، بخش «پارامتر نوع» ترجمه نمی‌شود
- در ترجمه، به جای «پارامتر نوع» خاص‌ترین نوع ممکن را جایگزین می‌کند
- به عنوان مثال، هر چهار خط زیر به یک شکل ترجمه می‌شوند
  - کد ترجمه‌شده‌ی هر چهار دستور زیر (در byte code) یکسان است

```
ArrayList<String> list = new ArrayList<>();
ArrayList<Integer> list = new ArrayList<>();
ArrayList<Object> list = new ArrayList<>();
ArrayList list = new ArrayList();//raw type
```

- البته قبل از اجرا (در زمان کامپایل) این متغیرها متفاوتند
- **کامپایلر** مراقب است شیء اول فقط با رشته‌ها فراخوانی شود، وگرنه: خطای کامپایل
  - مثلاً هنگام فراخوانی متد add روی این شیء، باید پارامتر از نوع رشته باشد

فرایند مَحو (ادامه)

```
class SortedList<T extends Comparable>{}
```

- مثال دیگر: کلاس SortedList فوق را در نظر بگیرید
- در ترجمه این کلاس، هر جا پارامتر نوع (T) استفاده شده،
  در هنگام ترجمه با Comparable جایگزین می‌شود
- همچنین کد ترجمه‌شده‌ی هر سه دستور زیر (byte code در) یکسان است

```
SortedList<Integer> list = new SortedList<>();
SortedList<String> list = new SortedList<>();
SortedList<Comparable> list = new SortedList<>();
```

- خلاصه: اِعمال محدودیت‌ها در زمینه داده‌های عام برعهده **کامپایلر** است
- بعد از کامپایلر (در زمان اجرا) محدودیتی در زمینه داده‌های عام اِعمال نمی‌شود

مثال

```
class Stack<T>{
   void push(T s){}
   T pop() {…}
}
```

ترجمه می‌شود به

```
class Stack{
   void push(Object s){}
   Object pop() {…}
}
```

```
class SortedList<T extends Comparable<T>>{
 T[] values;
 public void add(T o){…}
}
```

ترجمه می‌شود به

```
class SortedList{
   Comparable[] values;
   public void add(Comparable o){…}
}
```

```
static <E extends Comparable<E>> E max(E p1, E p2){…}
```

ترجمه می‌شود به

```
static Comparable max(Comparable p1, Comparable p2){…}
```

مثال از فرایند مَحو

- اگر کلاس روبرو را کامپایل کنیم:

```
public class Generic<T> {
 void f() {
   Generic<String> g1;
   g1 = new Generic<String>();
   Generic<Integer> g2;
   g2 = new Generic<Integer>();
 }
}
```

```
D:\java>javap -c Generic.class
Compiled from "Generic.java"
public class Generic<T> {
  public Generic();
    Code:
       0: aload_0
       1: invokespecial #1
       4: return

  void f();
    Code:
       0: new           #2
       3: dup
       4: invokespecial #3
       7: astore_1
       8: new           #2
      11: dup
      12: invokespecial #3
      15: astore_2
      16: return
}
```

- با کمک دستور javap (در JDK هست) می‌توانیم bytecode را ببینیم
- ایجاد دو متغیر g1 و g2 به یک شکل ترجمه شده است

کوییز

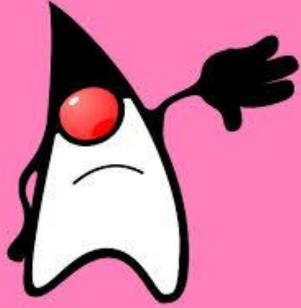
کوییز

- چرا نمی‌توانیم متد f را به شکل زیر سربار (overload) کنیم؟ (خطای کامپایل می‌گیریم)

```
public static <E extends Number> void f(E i){...}
public static void f(Number i){...}
```

- **پاسخ:**
- زیرا براساس فرایند مَحو، هر دو متد به یک شکل ترجمه می‌شوند
- پیغام کامپایلر:
- Erasure of method f(E) is the same as another method

محدودیت‌های داده‌های عام



محدودیت‌های انواع عام

- انواع داده عام، در جاوا محدودیت‌هایی دارند
- بسیاری از این محدودیت‌ها ناشی از فرایند محو (erasure) است
  - در زمان اجرا، «پارامتر نوع» استفاده شده در یک «شیء از نوع عام» معلوم نیست
  - مثلاً در زمان اجرا معلوم نیست که یک شیء از نوع List به شکل List<String> ایجاد (new) شده یا به شکل List<Integer>
  - به همین خاطر، در زمان اجرا، عملیات بر روی «پارامتر نوع» غیرممکن است
    - عملیاتی مثل new و instanceof که در زمان اجرا اتفاق می‌افتند
- مثال: نمونه‌سازی (با کمک new) از «پارامتر نوع» غیر ممکن است
  - مثلاً در کلاس Stack نمی‌توانیم با کمک new از نوع T یک شیء جدید بسازیم (خطای کامپایل)

```
class Stack<T>{
 T ref = new T();
}
```

فرض کنید:
```java
class Stack<T>{…}
```

محدودیت‌ها در تعریف انواع عام

- نمونه‌سازی از «پارامتر نوع» (T در کلاس فوق) ممکن نیست
  - ایجاد خطای کامپایل در داخل کلاس Stack ← `T ref = new T();`
- ایجاد (new) آرایه از پارامتر نوع ممکن نیست
  - خطای کامپایل ← `T[] elements = new T[size];`
- عملگر instanceof بر روی پارامتر نوع قابل فراخوانی نیست
  - خطای کامپایل: `if(o instanceof T);`
- تعریف متغیر استاتیک از جنس «پارامتر نوع» ممکن نیست
  - خطای کامپایل ← `private static T obj;`

محدودیت‌ها در استفاده از انواع عام

فرض کنید:

```
class Stack<T>{…}
```

- اگر پارامتر نوع را قید کنیم، ایجاد آرایه از نوع عام ممکن نیست
  - خطای کامپایل: `Stack<String>[] s = new Stack<String>[8];`
  - بدون خطا: `Stack[] t = new Stack[8];`
- استفاده از انواع داده اولیه (primitives) به عنوان پارامتر نوع ممکن نیست
  - خطای کامپایل ← `Stack<int> s;`
- در صورت تعیین پارامتر نوع، امکان instanceof برای نوع عام وجود ندارد
  - خطای کامپایل ← `if(o instanceof Stack<String>);`
  - بدون خطا ← `if(o instanceof Stack);`

نوع عام نمی‌تواند Exception باشد

- یک کلاس عام نمی‌تواند به عنوان استثنا (exception) استفاده شود
  - یک کلاس عام نمی‌تواند از Throwable ارث‌بری کند:

```
class GenExc<T> extends Exception {}
```
➔ syntax error

  - بنابراین شیئی از نوع عام را نمی‌توان پرتاب (throw) یا دریافت (catch) کرد
  - البته از «پارامتر نوع» (و نه از خود نوع عام) می‌توان به صورت استثنا استفاده کرد:

```
class Generic<T extends Exception> {
  void f() throws T {…}
  <E extends Throwable> void g() throws E {…}
}
```

کوییز

```java
1  public class Generics<T> {
2   void add(List<T> l, Object o) {
3    l.add((T) o);
4   }
5   public static void main(String[] args) {
6    Generics<String> g = new Generics<>();
7    List<String> list = new ArrayList<>();
8    list.add("a");
9    g.add(list, new Object());
10   g.add(list, new Integer(1));
11   for (String s : list){
12     System.out.println(s);
13   }
14  }
15 }
```

خروجی؟

الف) بدون خطا

ب) خطای کامپایل در خط ۹ و ۱۰

ج) خطای کامپایل در خط ۱۱ یا ۱۲

د) خطا در زمان اجرا در خط ۹

هـ) خطا در زمان اجرا در خط ۱۱

و) خطا در خط ۳

کوییز

- گفتیم امکان استفاده از انواع اولیه به عنوان پارامتر نوع نیست
- پس چرا این کد خطا ندارد؟ مگر ۵ یا num از انواع اولیه (int) نیستند؟

```
List<Integer> list = new ArrayList<>();
int num = 4;
list.add(num);
list.add(5);
```

- پاسخ:
- به دلیل **autoboxing**
- از جاوا ۵ به بعد، به صورت ضمنی مقدار 5 و num به Integer تبدیل می‌شوند
- به همان دلیل که این کد خطا ندارد: `Integer i = 3;`
- در واقع `list.add(5);` تقریباً معادل `list.add(new Integer(5));` است

جمع‌بندی

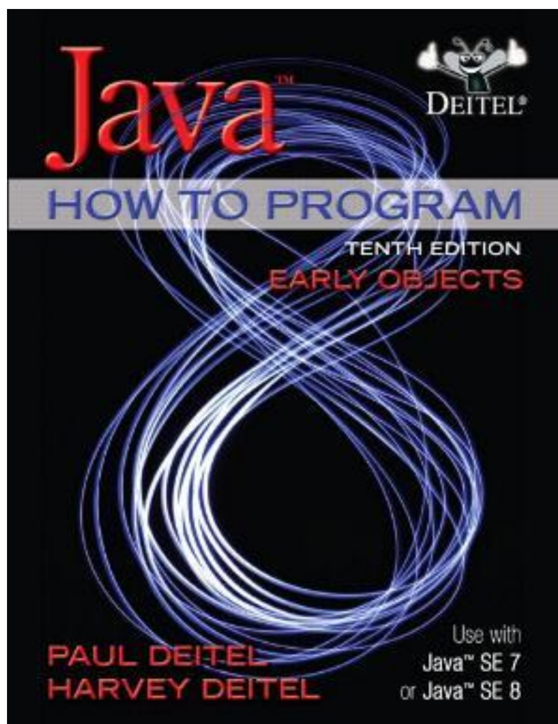
جمع‌بندی

- انواع داده عام و کاربردهای آن‌ها
  - مثال‌های موجود در زبان جاوا
- متدهای عام
- فرایند مَحو (Erasure)
- وراثت برای کلاس‌های عام
- محدودیت‌های انواع عام در جاوا

مطالعه کنید

- فصل ۲۰ و ۲۱ کتاب دایتل

Java How to Program (Deitel & Deitel)



20 Generic Classes and Methods 839

21 Custom Generic Data Structures 869

- تمرین‌های همین فصل‌ها از کتاب دایتل

تمرین

- واسط MyList را یک‌بار با کمک آرایه و بار دیگر با لیست‌پیوندی پیاده‌سازی کنید

```
interface MyList<E> {
  int size();
  boolean isEmpty();
  boolean contains(E o);
  boolean add(E e);
  boolean remove(E o);
  void clear();
  boolean equals(Object o);
  int indexOf(E o);
}
```

- این واسط مشابه java.util.List است
- درواقع باید سعی کنید کلاسی شبیه به ArrayList و کلاسی شبیه به LinkedList را پیاده کنید
- ابتدا سعی کنید، سپس نگاهی به متن کلاس‌های فوق بیاندازید

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

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

جستجو کنید و بخوانید

- موضوعات پیشنهادی برای جستجو:
  - مفهوم wildcards در انواع داده عام
  - زبان C++ دارای مفهوم Template به جای Generics است
    - C++ فرایند Erasure را پیاده نکرده، پس چگونه Template را ممکن کرده؟
  - زبان C# (در واقع، خانواده .NET) نیز دارای مفهوم Generics است
    - .NET هم فرایند Erasure را انتخاب نکرده، پس چگونه انواع عام را ممکن کرده؟
  - دلیل محدودیت‌های ذکر شده برای انواع عام در جاوا چیست؟
    - چرا بسیاری از این موارد در زبانی مثل C# وجود ندارد؟

پایان