PIC 20A
Generics

David Hyde
UCLA Mathematics

Last edited: May 15, 2020

Outline

Introductory example: ArrayList

`java.util.ArrayList<E>` is a *generic* class that represents a resizable array containing objects of type `E`.

```
import java.util.ArrayList;

public class Test {
  public static void main(String[] args) {
    ArrayList<String> s_arr =
                        new ArrayList<String>();

    s_arr.add("Hello");
    s_arr.add("World");
    s_arr.add("Java");
  }
}
```

`ArrayList<E>` of Java is like `std::Vector<T>` of C++.

Introductory example: ArrayList

The methods of `ArrayList<E>` are parameterized as well.
The method add has signature

```
public boolean add(E e)
```

This provides safety.

```
public static void main(String[] args) {
  ArrayList<String> s_arr =
                        new ArrayList<String>();

  s_arr.add(Boolean.FALSE);//error
}
```

Generic classes

classes can have parameterized types.

```
public class Pair<A,B> {
  private A first;
  private B second;

  public Pair(A first, B second) {
    this.first = first;
    this.second = second;
  }

  public void setFirst(A f) { first = f; }
  public void setSecond(B s) { second = s; }
  public A getFirst()   { return first; }
  public B getSecond() { return second; }
}
```

Generic interfaces

interfaces can have parameterized types.

```
public interface Comparable<T> {
  int compareTo(T o);
}
```

This is in fact the the java.lang.Comparable interface.

https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Writing generic methods

Methods can have parameterized types.

```
public class Util {
  public static <A, B>
   boolean samePair(Pair<A,B> p1, Pair<A,B> p2) {
    return (p1.getFirst().equals(p2.getFirst())
       && p1.getSecond().equals(p2.getSecond()));
  }
}
```

(`equals` is inherited from `Object`.)

Bounded type parameters

You can *bound*, i.e., set a requirement, for the type parameter.

```
public class Util {
  public static <T extends Comparable<T>>
                                T getMax(T a, T b) {
    if (a.compareTo(b) > 0)
      return a;
    else
      return b;
  }
}
```

When bounding, use `extends` for `classes` and `interfaces`.

Bounded type parameters

Type parameters must be bound in order to use members.
This is unlike templates in C++.

```
public class Util {
  public static <T> T getMax(T a, T b) {
    if (a.compareTo(b) > 0) //compile-time error
      return a;
    else
      return b;
  }
}
```

Since `T` is not bound, Java doesn't know whether `T` has a method named `compareTo`.

Enhanced for loop and interface Iterable

We can use the enhanced for loop with `Iterable` objects.

```
ArrayList<String> s_list;
s_list = new ArrayList<String>();
...
for (String s : s_list)
  System.out.println(s);
```

We can do this since `ArrayList<String>` implements the `interface java.lang.Iterable<String>`.

Enhanced for loop and interface Iterable

Under the hood, this is how the enhanced for loop works.

```
for (java.util.Iterator<String> iter
       = s_list.iterator(); iter.hasNext(); ) {
  String s = iter.next();
  System.out.println(s);
}
```

Raw type

When you omit the angled brackets, you get the *raw type*.

```
public static void main(String[] args) {
  ArrayList s_arr1 = new ArrayList();
  ArrayList<Object> s_arr2
                      = new ArrayList<Object>();
}
```

A raw type is similar to what you get when type parameter is `Object`. So `s_arr1` and `s_arr2` are almost the same, but the raw type is less safe.

Raw types should be avoided. I tell you about raw types only so that you don't accidentally use them. (Raw types are provided for backward compatibility. Generics were added to Java in version 5.)

Type inference

Often, you can omit the type parameter and let it be inferred.

```
public static void main(String[] args) {
  ArrayList<String> s_arr =
                    new ArrayList<>();
}
```

Type inference has nothing to do with using raw types.
You must use <> (the "diamond") to have the compiler infer the unspecified type parameter.

You can't use primitive types with generics

Primitive types are not objects. You can't do

```
public static void main(String[] args) {
  ArrayList<int> i_arr; //error
}
```