Generic Programming - Part 2

Chapter 8, Core Java Volume I

Contents

Bounds for Type Variables

- Sometimes, a type variable cannot be instantiated with arbitrary types:

```
class ArrayAlg
{
   public static <T> T min(T[] a)
   {
      if (a == null || a.length == 0) return null;
      T smallest = a[0];
      for (int i = 1; i < a.length; i++)
         if (smallest.compareTo(a[i]) > 0) smallest = a[i];
      return smallest;
   }
}
```

- How do we know that T has a `compareTo` method?

Bounds for Type Variables

- Need to restrict T in the method declaration:

```
public static < T extends Comparable<T> > T min(T[] a) ...
```

- Now min can be called with arrays of String, LocalDate, and so on, but not Rectangle.

- A type variable can have multiple bounds:

```
T extends Object & Comparable<T>
```

  - "extends" does not means T is a subclass of bounding classes, but a subtype.

Example: Testing Pair

```
import java.time.*;
public class PairTest2
{
   public static void main(String[] args)
   {
      String[] names =
         {
            “Hong”, “Kim”, “Ryu”, “Lee”
         };
      Pair<String> mm = ArrayAlg.minmax(names);
      System.out.println("min = " + mm.getFirst());
      System.out.println("max = " + mm.getSecond());
   }
}
```

```
class ArrayAlg
{
   public static <T extends Comparable<T>> Pair<T> minmax(T[] a)
   {
      if (a == null || a.length == 0) return null;
      T min = a[0];
      T max = a[0];
      for (int i = 1; i < a.length; i++)
      {
         if (min.compareTo(a[i]) > 0) min = a[i];
         if (max.compareTo(a[i]) < 0) max = a[i];
      }
      return new Pair<>(min, max);
   }
}
```

Type Erasure

- The Java Virtual Machine has no notion of generic types or methods.
- Generic classes and methods turn into ordinary classes and methods.
- Type variables are "erased", yielding a raw type.
- Type variables are replaced by their bounding types(or `Object` for variables without bounds)

```
public class Pair
{
   private Object first;
   private Object second;
   public Pair(Object first, Object second) { . . . }
   public Object getFirst() { return first; }
   public Object getSecond() { return second; }
   public void setFirst(Object newValue) { first = newValue; }
   public void setSecond(Object newValue) { second = newValue; }
}
```

Cast Insertion

- When your program calls to a generic method, the compiler automatically inserts casts when a return type has been erased:

```
Pair<Employee> buddies = . . .;              ⇒ Pair buddies = . . .;
Employee buddy = buddies.getFirst();   ⇒ Employee buddy = (Employee) buddies.getFirst();
```

- When you access a generic field, casts are inserted:

```
Employee buddy = buddies.first;    ⇒   Employee buddy = (Employee) buddies.first;
// suppose that the first field was public
```

- Casts are not needed for erased parameter types:

```
buddies.setFirst(buddy); // OK to convert Employee to Object
```

Calling Legacy Code

- When generics were added to Java, a major goal was to *interoperate with legacy code.*
- Example: Legacy class `Department` with methods

```
ArrayList getEmployees()
void addAll(ArrayList employees)
```

- Generic types can be implicitly casted to raw types:

```
ArrayList<Employee> newHires = . . .;
dept.addAll(newHires);
```

- Mixing generic types and raw types might generate warning:

```
 ArrayList<Employee> result = dept.getEmployees(); // warning
```

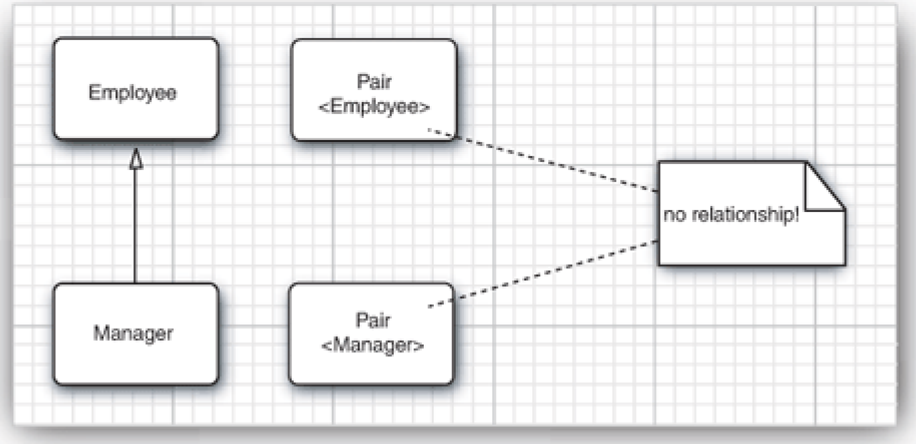
```
Note: PairTest1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Inheritance Rules for Generic Types

- Manager is a subclass of Employee. Is Pair<Manager> a subclass of Pair<Employee>?
- No subtype relationship between GenericType<Type1> and GenericType<Type2>.
  - Necessary for type safety:

```
Pair<Manager> managerBuddies = new Pair<>(ceo, cfo);
Pair<Employee> employeeBuddies = managerBuddies; // illegal, but suppose it wasn't
employeeBuddies.setFirst(lowlyEmployee);
```

  - What’s wrong?

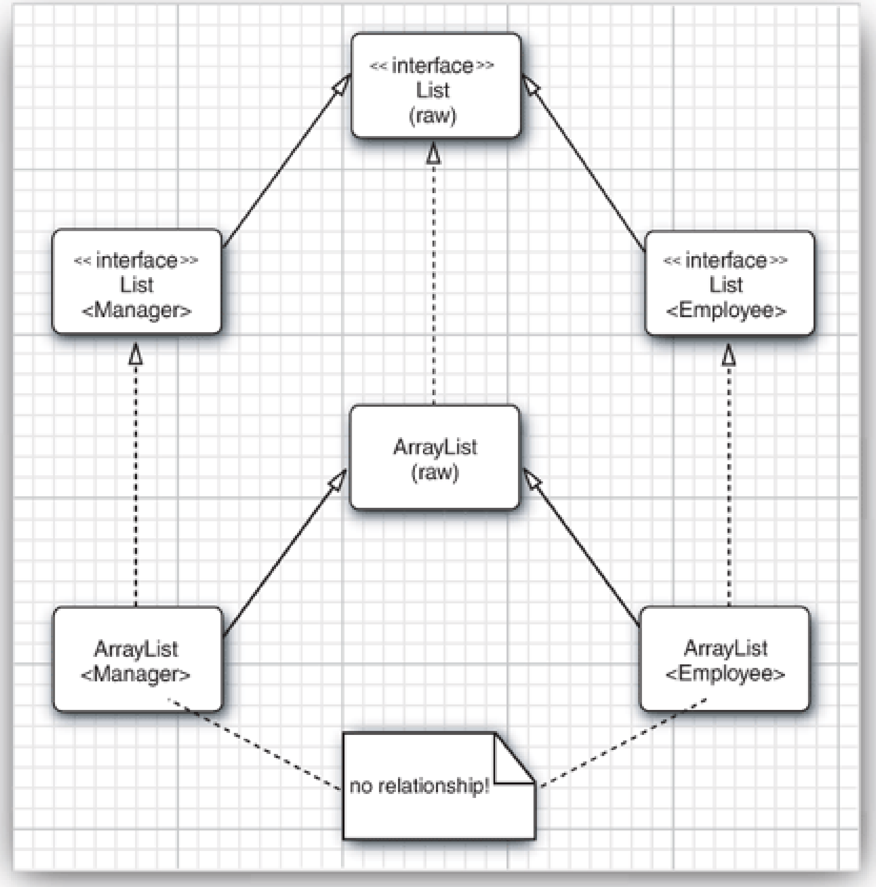


Inheritance Rules for Generic Types

- A parameterized type can be converted to a raw type.
  - Pair<Manager> is a subtype of the raw type Pair

```
Pair<Manager> managerBuddies = new Pair<>(ceo,cfo);
Pair rawBuddies = managerBuddies; // OK
rawBuddies.setFirst( new File("…") );
  // only a compile-time warning
```

  - When the foreign object is retrieved with getFirst and assigned to a Manager variable, a ClassCastException is thrown.



<< interface >> List (raw)
<< interface >> List <Manager>
<< interface >> List <Employee>
ArrayList (raw)
ArrayList <Manager>
ArrayList <Employee>
no relationship!