5. Collections and Generics

3. Generics

Generics Basics

- JDK 5.0 introduces generics
- Generics allow you to abstract over types
- The most common examples are container types, such as those in the Collections hierarchy

Generic Classes

public class *ClassName*<*T*>{

class body

}

- Parametric type *T* can be used in the class body as usual type:

```
private T a;
public void set(T a) { this.a = a; }
public T get() { return a; }
```

Generic Objects

- You need to set a type in <> when creating an object of generic class:

```
public class GenClass<T>{
    . . . . .
}

GenClass<Integer> cInt = new GenClass<Integer>();
```

How Generics Work

- In the invocation all occurrences of the formal type parameter are replaced by the *actual type argument*
- The compiler can check the type correctness of the program at compile-time
- Primitive types cannot use as actual types

Exercise. Print List

- Create a class with list of objects of an arbitrary given class with two methods:
  - add for accumulation data in the list
  - printList with a boolean parameter to print odd or even elements of the list accordingly to parameter's value

Exercise. Print List

- See 531FirstGeneric project for the full text

Generics Inheritance

- In general, if Sub is a subtype (subclass or subinterface) of Base, and G is some generic type declaration, it is **not** the case that G<Sub> is a subtype of G<Base>

Generic Interfaces

- Generic interfaces are similar to generic classes:

```
public interface List<E>{
    void add(E x);
    Iterator<E> iterator();
}
public interface Iterator<E>{
    E next();
    boolean hasNext();
}
```

Generic Methods

- Type parameters can also be declared within method and constructor signatures to create *generic methods* and *generic constructors:*

  public <U> void inspect(U u){ . . . }

- *Type inference* feature allows you to invoke a generic method as you would an ordinary method, without specifying a type between angle brackets

Generic Method Example

```
class ArrayAlg
{
   public static <T> T getMiddle(T[] a)
   {
      return a[a.length / 2];
   }
}
```

- You can define generic methods both inside ordinary classes and inside generic classes

Generic Method Call

- When you call a generic method, you can place the actual types, enclosed in angle brackets, before the method name:

```
String[] names = { "John", "Q.", "Public" };
String middle = ArrayAlg.<String>getMiddle(names);
```

Wildcards

- What **is** the supertype of all kinds of collections?
- Collection<Object> is not such supertype due to generics inheritance rule
- **Collection<?>** (pronounced "collection of unknown"), that is, a collection whose element type matches anything

Bounded Wildcards

- ? extends *class_name*
  - ? stands for an unknown type
  - that this unknown type is a subtype of *class_name*
  - *example:* List<? **extends** Shape>
- Code **<? super** *class_name* **>** would be read as "an unknown type that is a supertype of *class_name*, possibly *class_name* itself

Bounded Wildcards Example

```
public static double sumOfList(List<? extends
    Number> list) {
     double s = 0.0;
     for (Number n : list) s += n.doubleValue();
    return s;
}
```

Home Exercise 5.3.2 (1 of 2)

- Create TBill class that saves deal for buying treasury bills (nominal, price, amount of bills, maturity date) and calculating deal income as follows:

  income = (nominal – price) * amount

Home Exercise 5.3.2 (2 of 2)

- Create DealAnalisys class that saves deals of any type (depo – single, barrier, month capitalization, TBill)
- Create compareIncome method that compares yield of the deal that saved in the class object and deal given as method's parameter

Manuals

- http://docs.oracle.com/javase/tutorial/extra/generics/index.html