JAC444 - Lecture 2

Generics

Segment 4

Generics

In this segment you will be learning about:

- Generics in Java – Abstraction over Types

- Generic Methods and Bounded Type Parameters

- Wildcards and Subtyping

- Type Erasure

Simple Box Class

```java
/** Box class. */

public class Box {

    private Object o;

    public void add(Object o) {
        this.o = o;
    }

    public Object get() {
        return o;
    }
}
Box myBox = new Box();
myBox.add(new Integer(10));
Integer val = (Integer) myBox.get();
String str = (String) myBox.get();    java.lang.ClassCastException;
```

Defining Simple Generics

```
/** Generic Box class. */

public class Box<T> {                          ← generic type declaration

    private T t;                               ← formal type parameter

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}

  Box<Integer> integerBox = new Box<Integer>();
  Box<Integer> integerBox = new Box<>();
```

Type Parameter and Type Argument

- The generics can be used in: classes, interfaces, methods, constructors:

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

public interface Iterator<E> {
    E next();
    boolean hasNext();
}
```

- Generic type declaration `List<E>` is called *parameterized type*

  `E` in `List<E>` is called type parameter

- `List<Integer>` the formal type parameter `E` is replaced by the *actual type argument* `Integer`

  `Integer` in `List<Integer>` is called type argument

Multiple Type Parameters

```java
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}
public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Odd", 7);

OrderedPair<String, Box<Integer>> p =
    new OrderedPair<>("primes", new Box<Integer>(3));
```

Source: http://docs.oracle.com/javase/tutorial/java/generics/types.html

More Types

- Generic class can have multiple type parameters

- Type argument can be any user defined type

```
HashMap<String, Dog> map = new HashMap<String, Dog>();

    map.put("bliss" new Dog("myDog"));

    Dog d = map.get("bliss");
```

Generic method

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

- Type parameter's scope is limited to the method in which it is declared.

```
Public static <U> void inspect(U type) {

        System.out.println("Type parameter U is of Class: "
                + type.getClass().getName());

}
```

Bounded Type Parameters

- Restriction on the type parameter
  T is upper bounded by **Integer**

```
public class NaturalNumber<T extends Integer> {
    private T n;
    public NaturalNumber(T n)  { this.n = n; }
    public boolean isEven() {
        return n.intValue() % 2 == 0;
    }
 }
```

intValue() is the method from class **Integer**

Wildcards

- Consider the problem of writing a routine that shows out all the elements from a Box

```
void showAll ( Box<Object> b ) {
    for (Object o : b)
        System.out.println(o);
}
```

How could we invoke the method with `Box<String>` if `Box<String>` is not a subtype of a `Box<Object>`

We define `wildcard type` as `?` and a `Box<?>` as a Box of unknown types

```
void showAll ( Box<?> b ) {
    for (Object o : b)
        System.out.println(o);
}
```

Type Erasure

- When a generic type is instantiated, the compiler translates those types by a technique called *type erasure*

- `Box<String>` is translated to type Box, which is called the *raw type*

- When mixing legacy code with generic code, you may encounter warning messages similar to the following:

  ```
  Note: YourClass.java uses unchecked or unsafe operations.

  Note: Recompile with -Xlint:unchecked for details.
  ```

Generic and Raw Types

```
public class MixedClass {
    public static void main(String[] args){
            Box<Integer> bi;
            bi = createBox();
    }
    /* Pretend that this method is part of an old library, written
    before generics. It returns Box instead of Box<T>. */
    static Box createBox() {
            return new Box();
    }
}
```

```
MixedClass.java:4: warning: [unchecked] unchecked conversion
found : Box
required: Box<java.lang.Integer>
        bi = createBox();
                ^
1 warning
```

Upper/Lower Bounded Wildcards

- Upper/Lower-bounded wildcard is **?** character

Upper-bounded: `<? extends Number>`
means any type that is at least a **Number** type

Lower-bounded: `<? super Integer>`
means any type that is a super type of an **Integer** type

Source:
http://docs.oracle.com/javase/tutorial/java/generics/types.html