Generic Programming

Read Ch5

Outline

- Why do we need generic programming?
- Generic class
- Generic method
- Java interface, generic interface
- Java object type and conversion

Why do we need generic class?

- IntNode/DoubleNode/ByteNode/LocationNode/…
- IntArrayBag/DoubleArrayBag/ByteArrayBag/LocationBag/…

- A generic class (e.g., bag, node) can be used for all the above possible bags.

Outline

- Why do we need generic programming?
- Generic class
- Generic method
- Java interface, generic interface
- Java object type and conversion

Generic Class

```
public class ArrayBag<E> implements Cloneable{
        private E[ ] data;
        private int manyItems;
        public void add(E element){ //similar implementation as in IntArrayBag}
        …
}
```

Use a generic class:

```
ArrayBag<Integer> intbag = new ArrayBag<Integer>();
ArrayBag<String> strbag = new ArrayBag<String>();
intbag.add(4);
strbag.add(“Hello”);
```

Something to pay attention -- constructor

```
public ArrayBag(int initialCapacity)
{
    if (initialCapacity < 0)
      throw new IllegalArgumentException
      ("The initialCapacity is negative: " + initialCapacity);
    data = (E[]) new Object[initialCapacity];
    manyItems = 0;
}
```

data = new E[initialCapacity]; WRONG

Something to pay attention -- elements

- IntArrayBag
  - Data contains the real value of each element
- ArrayBag<E>
  - E[] data contains the reference to the real objects
- Example

- Implementations of Equals, countOfOccurences, search, etc.

```
for(int i=0; i<data.length;i++)
        if(data[i]==paramObj.data[i]
                …
for(int i=0; i<data.length;i++)
        if(data[i].equals(paramObj.data[i]);
                …
```

Something to pay attention – remove elements

- IntArrayBag
  - manyItems--;
- ArrayBag<E>
  - Set unused reference variables to null

Node, Linked List

- Cont…

Node<T>

```
public class Node<T>{
    private T data;
    private Node<T> link;
}
```

Outline

- Why do we need generic programming?
- Generic class
- Generic method
- Java interface, generic interface
- Java object type and conversion

How to write a general purpose class

- Generic programming
  - Template in C++
  - Parameterized types in Design patterns
- Solution 2: write Generic class

Generic method

```
Public <T> T getFirst(T[] data){
        if(data==null||data.length==0)
            return null;
        else
            return data[0];
}
```

T: Type

E: Element

Generic method

```
Public <T> T getFirst(T[] data){
        if(data==null||data.length==0)
            return null;
        else
            return data[0];
}

int i = getFirst (intArray);
String str = getFirst(strArray);
```

Generic type parameter
(1) ALWAYS appear in angle brackets right before the return type (for compiler)
(2) Used in parameter list
(3) Used like a class name

A compiler can detect certain type errors.

Some restrictions (p252)

- Cannot create a new array of elements in type T
  T[] tArray = new T[10]; WRONG
- Cannot call a constructor
  T tElement = new T(); WRONG

Outline

- Why do we need generic programming?
- Generic class
- Generic method
- Java interface, generic interface
- Java object type and conversion

Interfaces and iterators

- A java interface: primarily a list of related methods that a programmer may want to implement in a single class.
- In its most common form, an interface is a group of related methods with empty bodies.
- Interface implementation (use keyword implements)
  - Public class
  - If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.
- Example: people, student, staff

People interface

```
/**
 * Interface for a person
 */
public interface People {
      //Return the name of a person
      public String getName();

      //Return the age of a person
      public int getAge();

      public boolean equals(People p);

}
```

```
/**
 * Implement the People interface
 */
public class Staff implements People {

    private int age;
    private String name;

    public Staff(){age=0;name="";}
    public Staff(int page, String pName){age=page; name=pName;}

    @Override
    public String getName() {
        // return a silly name
        return "Non sense";
    }
    ...
}
```

```
/**
 * Implements the people interface
*/
public class Student implements People{
	private String name;

	@Override
	public String getName() {
		// return a fixed name
		return "Alice";
	}

	@Override
	public int getAge() {
		// return a fixed name
		return 0;
	}
	...
}
```

Interface as type

- A method use interface name as type
- The actual argument must be a data type that implements the interface

```
public class PeopleTest {

    public static void printPeople(People p){
        System.out.println(p.getName()+","+p.getAge());
    }

    public static void main(String[] args) {
        //People p = new People();//Wrong
        People s = new Student();
        People staff = new Staff();

        printPeople (s);
        printPeople (staff);
    }

}
```

Test whether a class implements an interface

- Instanceof

```
if(obj instanceof People)
    …
else
    …
```

Outline

- Why do we need generic programming?
- Generic class
- Generic method
- Java interface, generic interface
- Java object type and conversion

Generic interface

- A generic interface: specifies a list of methods, but these methods depend on one or more unspecified classes
  - Iterator<E>
  - Collection<E>
  - Comparable<T>
- A class implementing a generic interface is a generic class
- Example
  - public class MyCollection<E> implements Collection<E>,Cloneable
  - public class MyCollection<E> implements Collection<E>,Cloneable

Comparable<T> interface

- Class Location implements comparable<T>(Code)
- compareTo() method

- Example to use Location's compareTo method.
  - Different compareTo implementation

```
public class Location implements Comparable<Location>{

    private double x;
    private double y;

    public Location(double px, double py){x=px; y=py;}

    @Override
    //compare the locations first on x, then on y.
    public int compareTo(Location arg0) {
        Double objx = x;
        Double objy = y;
        if(objy.compareTo(arg0.y)!=0) return (objy.compareTo(arg0.y));
        else return (objx.compareTo(arg0.x));
    }
    …
}
```

Java Collection Interfaces

- Collection<E>
  - http://docs.oracle.com/javase/6/docs/api/java/util/Collection.html
- Java classes implementing Collection<E>
  - Vector, Set, List, ArrayList, SortedSet, HashSet
- Map interface (http://docs.oracle.com/javase/6/docs/api/java/util/Map.html)
  - TreeMap
  - HashMap

Collection and Iterator example

```
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class CollectionTest {
    public static void main(String[] args) {
        List<String> l1 = new ArrayList<String>(10);
        l1.add("method");  l1.add("is"); l1.add("bad");

        Iterator<String> iter = l1.iterator();
        int item=1;
        while(iter.hasNext()){
            String obj = iter.next();
            System.out.println((item++)+":"+obj);
        }
    }
}
```

How to write a general purpose class

- Solution 1: use Java Object type
- An object variable is capable of holding a reference to any kind of object.
  - int a;
  - Location b;
  - String c;
  - Object d;

Outline

- Why do we need generic programming?
- Generic method
- Generic class
- Java interface, generic interface
- Java object type and conversion

Widening conversions

String s = new String(“Hello world”);

Object obj = s;

Memory looks like?

- Given reference variables x and y, an assignment x=y is a widening conversion if the data type of x is capable of referring to a wider variety of things than the data type of y.

Narrowing conversions

String s = new String(“Hello world”);
Object obj = s;
String s2 = obj?
String s2= (String)obj;

Did you see such typecast anywhere?

- If a method returns an Object, then a narrowing conversion is usually needed to actually use the return value.

Primitive data type wrapper classes

- Boxing conversion
  - int i= 4;
  - Integer iobj = new Integer(i);
  - Integer iobj2 = i; //Autoboxing
- Unboxing conversion
  - int j = iobj.intValue();
  - int j2 = iobj; //Auto-unboxing

Use Object to implement general purpose class?

```
public class ArrayBag{
        private int manyItems;
        private Object[] data
        //constructoes….
        public static Object getFirst(){
                if(manyItems==0) return null;
                else return data[0];
        }
}
```

Calling function:

```
ArrayBag bag = new ArrayBag();
bag.add(new Integer(1));
…
//Get the first element
int fElement = (Integer) bag.getFirst();
```

Summary

- Object type
  - Widening/Narrowing conversion
- Primitive type
  - Wrapper class
  - Boxing, unboxing
- Generative method
- Generative classes
- Interface
  - How to write an interface, implement an interface, and use an interface
- Generative interface
  - Collection<E>, Iterator<E>, Comparable<E>
  - Get familiar with the classes in Java that implement these interfaces and know how to use some basic ones (e.g., ArrayList)

Reference

- Widening conversion, narrowing conversion, boxing, unboxing
  - Pages 253-257
- Generic method restriction
  - Page 260
- Generic class Lister
  - Page 289
- Comparable Generic Interface
  - Page 292