Computer Science 22: Object Oriented Programming

Lecture #13: Polymorphism

In This Lecture

- Types VS Class
- Polymorphism
- Kinds of Polymorphism
  - Parametric Polymorphism
    - Generics
  - Subtype/Inclusion Polymorphism
    - LSP
  - Ad hoc Polymorphism

Type VS Class

- The concept of **type** and **class** are different
- … although some treat these two terms as interchangeable (i.e., the **type** of an object is the same as the **class** of the object)

Type VS Class

- **Type** (as in **data type**)
  - Used in compile time for casting the type of an object thereby determining **what kinds of operations** are possible for the object
  - Used to ensure program correctness
- **Class**
  - A runtime determinant on the **implementation** of an operation
  - Used to determine the classification of the object

Type VS Class

```
public class Boxer {
    public void punch(){…}
    public void shout(){
          System.out.println(“Oof!”);
    }
}

public class KickBoxer extends Boxer {
    public void kick(){…}
    public void shout(){
          System.out.println(“Eych!”);
    }
}

Boxer bieber = new KickBoxer();
bieber.punch();
bieber.kick();
bieber.shout();
```

- **Type** of **bieber** is **Boxer**
- **Class** of **bieber** is **KickBoxer**

- **bieber.punch(); // valid**
- **bieber.kick(); //invalid**
- **bieber.shout(); //valid, ?**

Polymorphism

- Etymology: Greek: *poly* – **many**, *morphos* – **forms**
- In OOP, it is the…
  - Ability of an object to assume more than one type
  - Ability of an object to be used using a standard interface
    - As in the example, **`Boxer bieber = new KickBoxer;`**, **`bieber`** is used using the interface of **`Boxer`**

Kinds of Polymorphism

- **True Polymorphism**
  - **Parametric Polymorphism**
    - A method or a data type can be written generically so that it can handle values identically without depending on their types
  - **Inclusion/Subtype Polymorphism**
    - Subtypes can take the place of supertypes in methods requiring the supertype
- **Ad hoc Polymorphism**
  - Achieved through method overloading/overriding

Parametric Polymorphism

- In Java and C#, generics feature were added to allow parametric polymorphism in methods
- Consider a method for appending two lists:

```
public List append(List l1, List l2) {
   //returns the concatenated list
}
```

Under parametric polymorphism, it should be possible to design/implement this method without caring for the types of the objects contained in the list (i.e., list of strings, list of integers, etc.)

Java Generics

- Pre-Java 1.5 List

```
List myList = new LinkedList();
myList.add(“Sir Arian”);
String s = (String) myList.get(0);
```

- Before Java 5, the return types of List/Set/Map methods require you to typecast the object stored to its original class/type

Java Generics

- Java 5 and above

```
List<Integer> myList = new LinkedList<Integer>();
myList.add("Sir Arian");
String s = myList.get(0);
```

- List/Set/Map types can now be declared to indicate the kinds of objects to be stored (i.e., List of Strings, List of Integers, etc)

Java Generics

- Method parameters can now be more expressive

```
public List append(List l1, List l2) {…}
```

```
public List<String> append(List<String> l1,
  List<String> l2) {…}
```

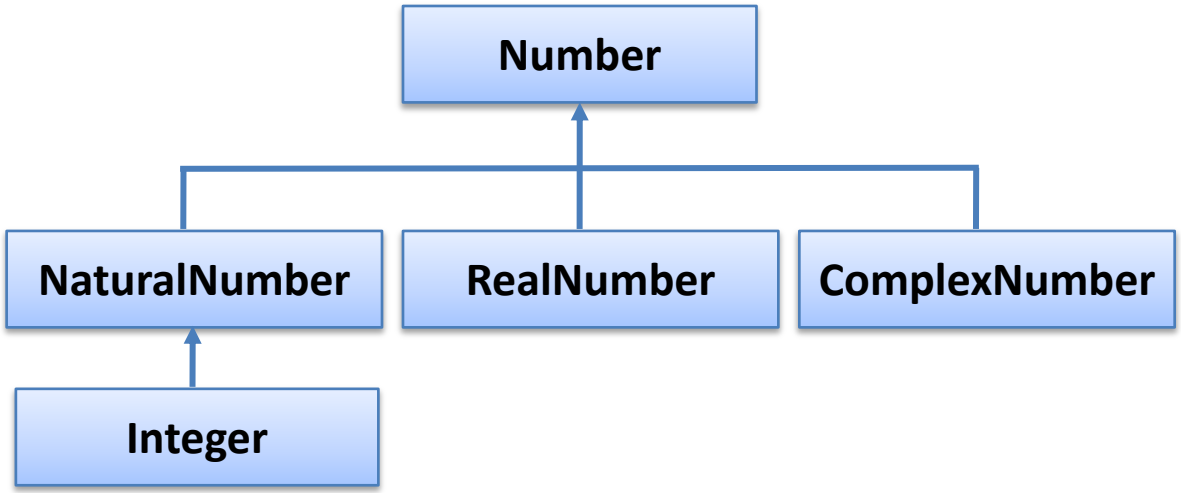
Subtype/Inclusion Polymorphism

- The Liskov Substitution Principle

Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T

The LSP

```
Number n = new Number(0);
Integer i = new Integer(10);
RealNumber r = new RealNumber(2.0);
ComplexNumber c = new ComplexNumber(2, 4); //2+4i
```



The LSP

- Suppose we have a method in Number:

```
public void multiply(Number n) {…}
```

Which of the following is valid?

```
n.multiply(i);
n.multiply(r);
n.multiply(c);
n.multiply(n);
c.multiply(n);
```

Ad hoc Polymorphism

- Achieved through method overloading/overriding
- Creating multiple methods with different parameters to give the user of the method the “feel” of using one method for any kind and number of parameters.

Ad hoc Polymorphism

```
//i.e., in java.lang.Math
Math.max(12, 13);           //two integers
Math.max(13f, 14.5f);       //two floats
Math.max(14d, 16d);         //two doubles
//etc
```