Alex Memering CM 2335
Eric Guilford CM 2221
Cameron Spry CM 2467

CSSE 221
Dr. Defoe

Polymorphism is a feature that allows different data types to use uniform methods. Instead of having separate methods named addInteger and addDouble to add these different types, Polymorphism gives one the ability to have one method that can add both of these data types without the need to specify the data type for the function. This could allow for tremendous amounts of variance that could be needed in different programs, being able to manipulate many different objects the same way even if they aren't the same type of objects. There are three separate types of Polymorphism, Ad-Hoc, Parametric and Subtyping; each allows one to write more flexible code that can handle a large variety of data types.

Ad-hoc Polymorphism (also known as overloading) makes a method able to operate on various types of parameters. One can make several methods with the same name in a class, with different parameter lists, and Java will call the proper one based on the parameters being passed to the method.

Example:
```
int doSomething(int a, int b) { }
int doSomething(double a, double b) { }
int doSomething(int a, int b, int c) { }

doSomething(5, 3) //calls first function
doSomething(3.4, 2.21) //calls second function
doSomething(5, 3, 2) //calls third function
```

Parametric Polymorphism (also known as Generic Programming) is much more general than Ad-Hoc; where in Ad-Hoc one must make a new method for each data type, in Parametric one makes a general method that does not care what data type is passed in. This obviously has its advantages, writing one method that will take care of each data type instead of writing each one individually. One simply has to specify the data type being handled at compile-time when the class containing the method is instantiated.

Example:
```
//The types handled by the lists are specified at compile-time
List<BigInteger> bigList = new ArrayList<BigInteger>();
List<Double> dblList = new ArrayList<Double>();

bigList.add(new BigInteger('45')); //bigList handles BigIntegers
dblList.add(new Double(5.421)); //dblList handles Doubles
```

The last type of Polymorphism is Subtype, which is the one that we are accustomed to with BigRational. In this type, if something is a subclass of another then the child object can be used safely in any context that was expecting the parent object.

Example:
```
void printObjString(Object a) {
      //all objects derive from the Object class, so they all
      //have toString().
      System.out.println(a.toString());
}

printObjString(new BigInteger('999')); //prints '999'
printObjString(new Double(43.1)); //prints '43.1'
```