CECS 277 – Lecture 13 – Generics

Generics allow you to specify the types of objects that a class or method will use. They use the type parameter to represent the types that will be passed in and used to construct the object. The type parameter is placed in angle brackets <> to specify it, and then used throughout the class whenever that type needs to be used.

Example: Class Declaration for a Generic Type

```
public class Pair <T>{
     private T a;
     private T b;
     public Pair( T x, T y ){
          setPair( x, y );
     }
     public T getA(){
          return a;
     }
     public T getB(){
          return b;
     }
     public void setPair( T x, T y ){
          a = x;
          b = y;
     }
}
```

Creating an instance of the Pair class is now the same as creating an ArrayList, the type is specified in angle brackets when declared.

```
Pair <String> p = new Pair <String> ("Cat", "Meow");

Pair <Integer> n = new Pair <Integer> ( 3, 3 );

Color r = new Color (Color.RED);
Color g = new Color (Color.GREEN);
Pair <Color> c = new Pair <Color> ( r, g );

Pair s = new Pair( "Apple", .67 );
```

Each of the above lines are creating a pair of objects of different types. All but the last line are creating pairs of the same type. This line will return a warning by the compiler, but it will still run. This is because the compiler will give the default type of Object to both of these values, effectively making them the same type. But now, if there are any type specific manipulations or methods that were needed to be used on those objects, they will need to be downcasted to those types first. It is also possible to declare multiple generic types if necessary.

```
public class Pair <S, T>{
```

Passing Generic Types to Methods –

If you know what type a generic should be when you pass it to, or return from a method, then use that type, but if it is possible that you will not know what types will be in the generic, then you can use a wild-card type instead by using a `?`.

Example: Method passing in a Generic with a Defined Type

```
public static int multPair ( Pair <Integer> p ){
     return p.getA() * p.getB();
}
```

Example: Method passing in a Generic with a Wild-Card Type

```
public static void displayPair ( Pair <?> p ){
     System.out.println("A = " + p.getA());
     System.out.println("B = " + p.getB());

}
```

Wild-Cards can be constrained so that not just any type can be used. They can use the inheritance chain to give an upper or lower bound type. An upper bound can be created by using the keyword extends, and a lower bound can be created using super.

Example: Method passing in a Generic with a Wild-Card Type with an Upper Bound

```
public static void animPair(Pair<? extends Animal> p){
     System.out.println("Animal1 = " + p.getA());
     System.out.println("Animal2 = " + p.getB());

}
```

Generic Methods – Similar to classes, methods can also be made to be generic.

```
public static void displayArr( T [] arr ){
     for(int i=0;i<arr.length;i++){
          System.out.print(arr[i]+ " ");
     }
     System.out.println();
}
```

Just like before, generic methods can also be constrained by using upper and lower bounds, and can take in multiple generic types.