Generic Methods

…

What are Generic Methods?

Generic methods = methods that introduce type parameters

- Similar to declaring a generic type
- but type parameter's scope is limited to method where it is declared

The following are allowed

- Static and non-static generic methods
- generic class constructors

Consider this…

```
public class Util {

    public static <K, V> boolean compare(
            Pair<K, V> p1,
            Pair<K, V> p2)
    {
      return
            p1.getKey().equals(p2.getKey())
              &&
            p1.getValue().equals(p2.getValue())
      ;
    }
}
```

These are the Pair Objects we want to compare(…)

```
public class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key    = key;
        this.value  = value;
    }

    public void     setKey(K key)         { this.key = key; }
    public void     setValue(V value)     { this.value = value; }
    public K        getKey()              { return key; }
    public V        getValue()            { return value; }
}
```

How do we use this static compare method?

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");

boolean same = Util.<Integer, String>compare(p1, p2);
```

This is the syntax to specify the Explicit Type Parameters

? Is it necessary to specify the actual type parameters for the methods for each invocation?

Type Inference to the rescue!

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");

boolean same = Util.<Integer, String>compare(p1, p2);
```

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");

boolean same = Util.compare(p1, p2);
```

What is Type Inference?