Generics

CS 2833 Modular Software Development

Department of Computer Science and Engineering

University of Moratuwa

Generics

- Generic programming involves the design and implementation of code that work for multiple data types.
- This is a way to satisfy compile time type checking for more stable code where the wrong data type cannot be casted.
- Eliminates casting,

G**eneric Methods** – Can use same generic method declaration that can be called with different types of arguments.
Type parameter is indicated in angle brackets.
It can contain one or more type parameters separated by commas.

```
public class MyClass {

   public static < E > void printArray( E[] inputArray ) {
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }
```

Generic Classes – Class name is followed by the type parameter section.
These classes are also known as parameterized classes.

```
public class MyClass <E> {
   private E e;

   public void doSomething(E e) {
      this.e = e;
   }

}
```

Generics

- This provides a way to re-use the same code with different inputs.

```java
public class GenericMethod {

    public <E> void printArray (E[] inputArray) {

        for (E element: inputArray) {
            System.out.printf("%s ", element);
        }

        System.out.println("Completed one type");
    }

    public static void main(String[] args) {

        GenericMethod g = new GenericMethod();

        Integer[] intArray = {1,2,3,4,5};
        String[] stringArray = {"A", "B", "C", "D"};

        g.printArray(intArray);
        g.printArray(stringArray);

    }

}
```

Parameter Namin Convention

- Type parameter names are single, uppercase letters:
  - T - Type
  - E - Element
  - K - Key
  - N - Number
  - V - Value

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {

    private K key;
    private V value;

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

    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```

```
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
Pair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");
```

Example

```
class Apple {

    private static long counter;
    private final long id = counter++;

    public long id() {
        return id;
    }
}

class GrannySmith extends Apple {}

class Gala extends Apple {}

class Fuji extends Apple {}

public class GenericAndUpcasting {

    public static void main(String[] args) {

        ArrayList<Apple> apples = new ArrayList<Apple>();
        apples.add(new GrannySmith());
        apples.add(new Gala());
        apples.add(new Fuji());
        apples.add(new GrannySmith());

        for (Apple c : apples) {
            System.out.println(c);
        }
    }
}
```