Generics. Generic Classes and Generic Methods

1. Java Generics

Motivation:

- to declare a single method which works with different data types
- to declare a single class which represents a set of related types

Generic types and methods are general purpose types and methods that operate with `Object` data type i.e. with **any data type** (e.g. a sort method that can sort integers, doubles, strings, etc.) and have **type parameters**.

The term Generics refers to language features related to the definition and use of generic types and methods.

Type parameters naming convention:

- T – Type
- E – Element
- K – Key
- N – Number
- V – Value

2. Application

- When grouping data, usually a **homogenous** collection is needed, i.e. which contains elements of the same type. Most Java Collection Framework classes take parameters of `Object` type and return values as `Object`, as well. Generics features allow you to use the framework with **specific type of objects**.
- Generics provide compile-time type safety - allow programmers to **catch invalid types at compile time**:

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(100);
numbers.add("test");
```

```
Error: The method add(java.lang.Integer) in the type
java.util.ArrayList<java.lang.Integer> is not applicable
for the arguments (java.lang.String)
```

- **Casting is not required** when processing items from a collection:

```
// casting required for general objects
ArrayList objectList = new ArrayList();
objectList.add("casting required");
String s1 = (String)objectList.get(0); //type casting
System.out.println(s1);
// applying Generics: no need to cast the object taken from the collection
ArrayList<String> specificList = new ArrayList<String>();
specificList.add("casting not required");
String s2 = specificList.get(0);
System.out.println(s2);
```

3. Generic Methods

Generic methods are methods that introduce their own type parameters. Parameter's scope is limited to the method where it is declared. Static and non-static generic methods are allowed, as well as generic class constructors:

type parameter (one or more types, separated by commas)

```
// generic method
public static <E> void printArray(E[] items){
    for (E item: items){
        System.out.println(item);
    }
    System.out.println();
}
```

placeholder for the type of the argument(s)

```
// using a generic method
Integer[] intArray = {1,2,3,4,5};
printArray(intArray);
Character[] charArray = {'a','b','c','d','e'};
printArray(charArray);
```

4. Generic Classes

A generic class declaration looks like a non-generic class declaration, except that the **class name is followed** by a type parameter section:

```
//user-defined generic class
public static class SimpleGeneric<T>{
    private T objReff = null;
    //the constructor accepts type parameter T
    public SimpleGeneric(T param){
        this.objReff = param;
    }
    public T getObjReff(){
        return this.objReff;
    }
    //this method prints the instance variable type
    public void printType(){
        System.out.println("Type: "+ objReff.getClass().getName());
    }
}
```

```
//using a generic class
SimpleGeneric<String> school = new SimpleGeneric<String>("RHHS");
school.printType();
SimpleGeneric<Integer> count = new SimpleGeneric<Integer>(1750);
count.printType();
```

```
//user-defined generic class with two type parameters
public static class TwoParGeneric<U, V>{
    private U objUreff;
    private V objVreff;
    //the constructor accepts object type U and object type V
    public TwoParGeneric(U objU, V objV){
        this.objUreff = objU;
        this.objVreff = objV;
    }
    //this method prints the instance variables types
    public void printTypes(){
        System.out.println("U Type: "+this.objUreff.getClass().getName());
        System.out.println("V Type: "+this.objVreff.getClass().getName());
    }
}
```

```
//using a generic class with two type parameters
TwoParGeneric<String,Double> item = new TwoParGeneric<String,Double>("gasoline",1.39);
item.printTypes();
```

Remember that generics only works on objects, not primitive types!