Course code : **CSE2005**

Course title : **Object Oriented Programming**

Generics

Objectives

This session will give the knowledge about

- Generics

What Are Generics?

- The term generics means parameterized types.
- Using generics, it is possible to create a single class, for example, that automatically works with different types of data.
- A class, interface, or method that operates on a parameterized type is called generic, as in generic class or generic method.
- The pre-generics code, generalized classes, interfaces, and methods used Object references to operate on various types of objects. The problem was that they could not do so with type safety.

What are and Why Generics?

Mechanism by which a single piece of code can manipulate many different data types without explicitly having a separate entity for each data type

```
//before Generics
List lt=new ArrayList();
lt.add(23);
lt.add(45.43);
```

```
//after Generics
List<Integer> lt1=new ArrayList<Integer>();
lt1.add(23);
lt2.add(45.43);
```

Generics added the type safety that was lacking. They also streamlined the process type casting which leads type cast error.

What problems does Generics solve?

Problem: Collection element types

- Compiler is unable to verify types
- Assignment must have type casting
- ClassCastException can occur during runtime

Solution: Generics

- Tell the compiler type of the collection
- Let the compiler fill in the cast
- Example: Compiler will check if you are adding Integer type entry to a String type collection (compile time detection of type mismatch)

General Form of a Generic Class

The syntax for declaring a generic class:

class class-name<type-param-list > { // …

Here is the full syntax for declaring a reference to a generic class and instance creation:

class-name<type-arg-list > var-name

= new class-name<type-arg-list >(cons-arg-list);

A Simple Generics Example

```
import java.util.*;
public class GenericDemo {
        public static void main(String[] args) {
                Set<Integer> set=new HashSet<Integer>();
                set.add(23);    set.add(45);
                set.add(12);    set.add(23);
                Iterator<Integer> it=set.iterator();
                int sum=0;
                while(it.hasNext())
                        sum+=it.next();
                System.out.println(sum);
        }
}
```

A Simple Generics Example

```
class Mark<Type>{
        Type obj;
        Mark(Type obj){
                this.obj=obj;
        }
        public Type getObj() {
                return obj;
        }
        public void setObj(Type obj) {
                this.obj = obj;
        }
```

A Simple Generics Example

```
        public String toString(){
                return obj.getClass().getName();
        }
}
public class Main {
        public static void main(String[] arg) {
                Mark<Integer> sem1;
                sem1=new Mark<Integer>(77);
                System.out.println("class Mark is of type:"+sem1);
                System.out.println(sem1.getObj());
```

A Simple Generics Example

```
                    Mark<Float> sem2;
                    sem2=new Mark<Float>(77.45f);
                    System.out.println("class Mark is of type:"+sem2);
                    System.out.println(sem2.getObj());
          }
}
```

Generics Work Only with Reference Types

```
public static void main(String[] arg) {
        //Correct
        Mark<Integer> sem1=new Mark<Integer>(77);
        System.out.println("class Mark is of type:"+sem1);
        System.out.println(sem1.getObj());

        //Wrong
        Mark<int> sem1=new Mark<int>(77);
        System.out.println("class Mark is of type:"+sem1);
        System.out.println(sem1.getObj());
    }
```

Generic Types Differ Based on Their Type Arguments

```
public static void main(String[] arg) {
        Mark<Integer> sem1=new Mark<Integer>(77);
        System.out.println("class Mark is of type:"+sem1);
        System.out.println(sem1.getObj());

        Mark<Float> sem2=new Mark<Float>(77.45f);
        System.out.println("class Mark is of type:"+sem2);
        System.out.println(sem2.getObj());

        //Wrong
        sem1=sem2;
}
```

How Generics Improve Type Safety

```
class Mark{
        Object obj;
        Mark(Object obj){
                this.obj=obj;
        }
        public Object getObj() {
                return obj;
        }

        public void setObj(Object obj) {
                this.obj = obj;
        }
```

How Generics Improve Type Safety

```
		public String toString(){
				return obj.getClass().getName();
		}
}

class Grade<Type>{
		Type obj;
		Grade(Type obj){
				this.obj=obj;
		}
```

How Generics Improve Type Safety

```
        public Type getObj() {
                return obj;
        }

        public void setObj(Type obj) {
                this.obj = obj;
        }

        public String toString(){
                return obj.getClass().getName();
        }
}
```

How Generics Improve Type Safety

```
public class Main {
        public static void main(String[] arg) {
                Mark sem1=new Mark(23);
                sem1.setObj(24.45f);

                Grade<Integer> grad1=new Grade<Integer>(77);
                grad1.setObj(75.45); //here Generic ensures Type safety
        }
}
```

A Generic Class with Two Type Parameters

```
public class GenericDemo {
    public static void main(String[] args) {
        Map<Character, Integer> count = new HashMap<Character, Integer>();
        String inp = "i am java";

        for (char c : inp.toCharArray()) {
            if (count.containsKey(c)) {
                count.put(c, count.get(c) + 1);
            } else {
                count.put(c, 1);
            }
        }
```

A Generic Class with Two Type Parameters

```
            for (Map.Entry entry : count.entrySet()) {
                System.out.println(entry.getKey() + " occurs " + entry.getValue());
            }

            //Set salary = map.entrySet();
            //Iterator it = salary.iterator();
            //while(it.hasNext()){
            //Map.Entry e = (Map.Entry) it.next();
            //System.out.println(e.getKey()+" : "+e.getValue());
        }
    }
```

A Generic Class with Two Type Parameters

```
class Student<Regno,Rank>{
        Regno rno;
        Rank rank;
        public Student(Regno rno, Rank rank) {
                this.rno = rno;
                this.rank = rank;
        }
        public Regno getRno() {
                return rno;
        }
        public void setRno(Regno rno) {
                this.rno = rno;
```

A Generic Class with Two Type Parameters

```
          }
          public Rank getRank() {
                    return rank;
          }
          public void setRank(Rank rank) {
                    this.rank = rank;
          }
          public String toString() {
                    return "Student [rno=" + rno + ", rank=" + rank + "]";
          }
}
```

A Generic Class with Two Type Parameters

```
public class Main {
    public static void main(String[] arg) {
        Student<String,Integer> s1=new Student<String,Integer>("19bcd7112",2);
        System.out.println(s1);
        Student<Integer,String> s2=new Student<Integer,String>(1,"19bcd7122");
        System.out.println(s2);
    }
}
```

Summary

We have discussed about

- Generics