Generics

Generics are concept launch by java in JDK 1.5 version and the major goal of generic is to avoid the ClassCastException at run time.

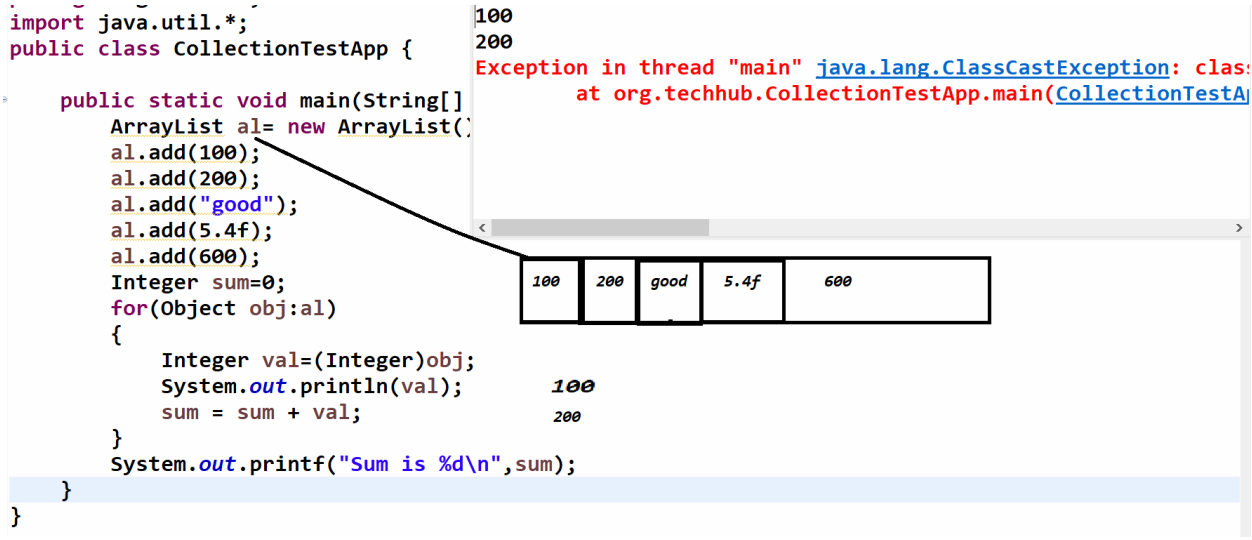
Q. What is the ClassCastException

ClassCastException means when we perform type casting between two different objects with each other but when object type or class type not matches with each other then ClassCastException occur at program time.

Q. Can you Give RealTime Example Where ClassCastException occur?

If we use the Collection then we have the benefit of Collection is we can store the any type of data in it but sometime this behavior of collection may be generate the ClassCastException at run time . Because Collection return every element in the form Object class and in java Object class is only use for store the data we cannot perform any operation on Object class and if we want to perform any Operation on Object class we have to type the cast the Object in its original type so in this case there is possibility to generate the ClassCastException at run time and if we want to resolve this problem we have the manage the generics with Collection.

Example



If we think about above code we store the different type of data in ArrayList collection and we try to fetch it and try to convert in Integer type from Object type so first two values i.e. 100 and 200 get converted because their original type is Integer so we can case Object class to Integer property we have the third element value is good and its original type String and we cannot cast String type to Integer type directly so we get error at run time called as ClassCastException

How to use the Generics in java

If we want to use the generics in java we have the following syntax < > this is the notation of Generics

Example: ArrayList <Integer> ref = new ArrayList<Integer> ();

We apply the compile time restriction on ArrayList as per our Example means if we try to store the other data in ArrayList than integer then it will generate the error to us at compile time.

Source code

```
package org.techhub;
import java.util.*;
public class GenApplication {
```

```
    public static void main(String[] args) {
        ArrayList <Integer>al = new ArrayList<Integer>();
                al.add(100);
                al.add(200);
                al.add(300);
                al.add(400);
                int sum=0;
                for(Integer val: al)
                {   sum  = sum  + val;
                }
                System.out.println("Sum is  "+sum);
        }
}
```

If we think about above code then there is limitation ArrayList hold only integer value but if we required the String, integer and Float type in ArrayList. In this case we can use the POJO class store the all data in POJO class and store the POJO class objects in collection.

Example

```
package org.techhub;
import java.util.*;
class Data
{
        private String name;
        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public int getId() {
                return id;
```

```
	}

	public void setId(int id) {
		this.id = id;
	}

	public float getPer() {
		return per;
	}

	public void setPer(float per) {
		this.per = per;
	}

	private int id;
	private float per;

	public Data(String name,int id,float per)
	{
		this.name=name;
		this.id=id;
		this.per=per;
	}

}
public class GenApplication {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ArrayList <Data>al = new ArrayList<Data>();
		Data d1=new Data("Ram",1,90);
		Data d2 =new Data("Shyam",2,80);
		Data d3=new Data("Dinesh",3,70);
		Data d4=new Data("Rajesh",5,60);
		al.add(d1);
		al.add(d2);
		al.add(d3);
		al.add(d4);
```

```
            for(Data d:al)
            {
             System.out.println(d.getId()+"\t"+d.getName()+"\t"+d.getPer());
            }
        }
}
```

Output

```
1       Ram     90.0
2       Shyam   80.0
3       Dinesh  70.0
5       Rajesh  60.0
```

Types of Generics

1) Class with Generics

2) Interface with Generics

3) Wild card generics

   a) Bounded generics

   b) Unbounded generics

If we want to work with user defined generics we have the some important notation provided by java to us.

E: Generic Element normally we refer this notation when we want to use the single value as Generic

T: Generic Type element this is normally use for when we want to use the Generic Array

K: Generic Key value normally this notation is used for map with key

V: Generic value normally this notation is used for map with value.

All notations are inbuilt classes in java

Class With generic concept

We can create your own class as generic class in java so if we want to declare your own class as generic class we have following syntax.

```
class classname<GenericNotation>{

  returntype functionname(GenericNotation ref) {

  }

}
```

Example

```
package org.techhub;
class A<E>
{
        void setValue(E e)//E can store any type of data but we can apply
restriction on using generics
        {   System.out.println("E is "+e);
  }
}
public class OwnGenApplication {
        public static void main(String[] args) {
                A <Integer>a1 =new A<Integer>();
                a1.setValue(100);
                a1.setValue(600);
                a1.setValue(500);
        }
}
```

Generic with Interface

If we use the generics with interface we have the major benefit is we can customize the method parameter type or change method parameter type in every implementer class.

Example

```
package org.techhub;
interface Area<E>
{
	void setRadius(E e);
}
class Circle<E> implements Area<Integer>
{	@Override
	public void setRadius(Integer e) {
		// TODO Auto-generated method stub
		System.out.println("I am integer radius "+e);
	}

}
class Cirm<E> implements Area<Float>
{
   @Override
	public void setRadius(Float e) {
		System.out.println("I am float Radius "+e);
	}
}
public class GenericWithInterfaceApplication {
	public static void main(String[] args) {
	Circle <Integer> c = new Circle<Integer>();
	c.setRadius(5);
	Cirm cm = new Cirm();
	cm.setRadius(5.6f);

	}

}
```