SNS COLLEGE OF ENGINEERING

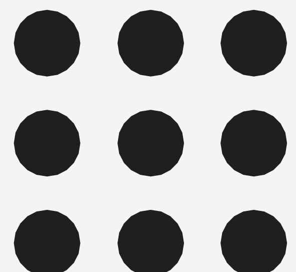
Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Information Technology

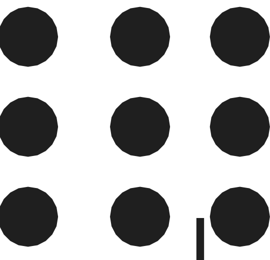
19CS204 OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Topic – Generic Programming



Generic Programming



- The term generics means parameterized types.
- Parameterized types enable you to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter.
- The Java Generics allows us to create a single class, interface, and method that can be used with different types of data.
- 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.

Generic Class

Example

```
class Gen<T> { //T is type parameter
T ob; // declare an object of type T
// Pass the constructor a reference to
// an object of type T.
Gen(T o) {
ob = o;
}
// Return ob.
T getob() {
return ob;
}
// Show type of T.
void showType() {
System.out.println("Type of T is " + ob.getClass().getName());
}
}
```

```
class GenDemo {
public static void main(String args[]) {
// Create a Gen reference for Integers.
Gen<Integer> iOb;
iOb = new Gen<Integer>(88);
// Show the type of data used by iOb.
iOb.showType();
int v = iOb.getob();
System.out.println("value: " + v);
System.out.println();
// Create a Gen object for Strings.
Gen<String> strOb = new Gen<String> ("Generics Test");
// Show the type of data used by strOb.
strOb.showType();
String str = strOb.getob();
System.out.println("value: " + str);
}
}
```

Generic Class

Example

```
class TwoGen<T, V> {
T ob1;
V ob2;
TwoGen(T o1, V o2) {
ob1 = o1;
ob2 = o2;
}
// Show types of T and V.
void showTypes() {
System.out.println("Type of T is " + ob1.getClass().getName());
System.out.println("Type of V is " + ob2.getClass().getName());
}
T getob1() {
return ob1;
}
V getob2() {
return ob2;
}
}
```

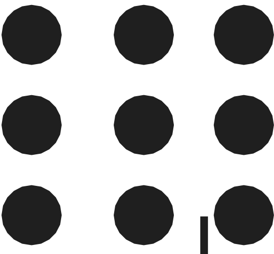
```
class SimpGen {
public static void main(String args[]) {
TwoGen<Integer, String> tgObj =
new TwoGen<Integer, String>(88, "Generics");
// Show the types.
tgObj.showTypes();
// Obtain and show values.
int v = tgObj.getob1();
System.out.println("value: " + v);
String str = tgObj.getob2();
System.out.println("value: " + str);
}
}
```

Generic – Bounded Types

- There may be times when you'll want to restrict the kinds of types that are allowed to be passed to a type parameter.
- For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what bounded type parameters are for.
- To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its upper bound.
- For example, If you want a generic class that works only with numbers (like int, double, float, long …..) then declare type parameter of that class as a bounded type to java.lang.Number class
- Here is the syntax for declaring Bounded type parameters.

  <T extends SuperClass>

  This specifies that ‘T’ can only be replaced by ‘SuperClass’ or it’s sub classes.

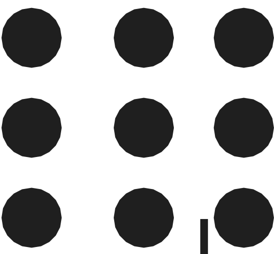


Generic – Bounded Types

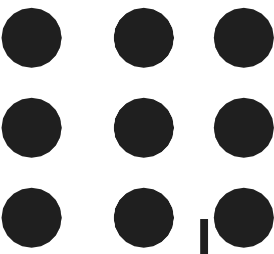
```
class Stats<T extends Number> {
T[] nums; // array of Number or subclass
Stats(T[] o) {
nums = o;
}
double average() {
double sum = 0.0;
for(int i=0; i < nums.length; i++)
sum += nums[i].doubleValue();
return sum / nums.length;
}
}
```

```
class BoundsDemo {
public static void main(String args[]) {
Integer inums[] = { 1, 2, 3, 4, 5 };
Stats<Integer> iob = new Stats<Integer>(inums);
double v = iob.average();
System.out.println("iob average is " + v);
Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
Stats<Double> dob = new Stats<Double>(dnums);
double w = dob.average();
System.out.println("dob average is " + w);
//String strs[] = { "1", "2", "3", "4", "5" };
// Stats<String> strob = new Stats<String>(strs);
// double x = strob.average();
// System.out.println("strob average is " + v);
}
}
```

Generic Methods



- It is possible to declare a generic method that uses one or more type parameters of its own.
- Furthermore, it is possible to create a generic method that is enclosed within a non-generic class.
- It allows static as well as non-static methods.
- Here, the scope of arguments is limited to the method where it is declared



Generic Methods

```
class GenMethDemo {
// Determine if an object is in an array.
static <T extends Comparable<T>,
V extends T> boolean isIn(T x, V[] y) {
for(int i=0; i < y.length; i++)
if(x.equals(y[i])) return true;
return false;
}
```

```
public static void main(String args[]) {
// Use isIn() on Integers.
Integer nums[] = { 1, 2, 3, 4, 5 };
if(isIn(2, nums))
System.out.println("2 is in nums");
if(!isIn(7, nums))
System.out.println("7 is not in nums");
System.out.println();
// Use isIn() on Strings.
String strs[] = { "one", "two", "three",
"four", "five" };
if(isIn("two", strs))
System.out.println("two is in strs");
if(!isIn("seven", strs))
System.out.println("seven is not in strs");
// Oops! Won't compile! Types must be compatible.
// if(isIn("two", nums))
// System.out.println("two is in strs");
}
}
```

THANK YOU