1

Generics in Java

The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects.

It makes the code stable by detecting the bugs at compile time. Generics enable programmer to

define a template (or common or generic) like class (or method or interface) that can work in

common with more than one type that means it increases code reusability in a program. Generics

in Java are something like templates in C++. These are actually typed classes (or interfaces or

methods). Type parameter naming convention is caps letter starting from T, U, and V etc.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now

generics force the java programmer to store a specific type of objects that is type-safe

programming.

Example of non-generic versus generic classes

Non-generic class having integer member:

class c1

{

int x;

c1(int v)

{x=v;}

void f1()

{ System.out.println(x);}

}

class c123

{

public static void main(String args[])

{

c1 ob1=new c1(10);

ob1.f1();

}

}

2

Non-generic class having string member:

class c2

{

String x;

c2(String v)

{x=v;}

void f2()

{ System.out.println(x);}

}

class c234

{

public static void main(String args[])

{

c2 ob2=new c2(“Hello123”);

ob2.f2();

}

}

Likewise, without generics to do same work on more than one type, we need to define

more than one classes. But, to define a template like common solution we use generic class.

Generic class having typed member that will be decided at run-time while creating object

via angle brackets:

class c2<T>

{

T x;

c2(T v)

{x=v;}

void f2()

{ System.out.println(x);}

}

class c456

{

public static void main(String args[])

3

{

c2<Integer> ob2=new c2<Integer>(10);

ob2.f2();

c2<String> ob3=new c2<String>("Hello Generics");

ob3.f2();

//Note: works only with reference types i.e. wrapper classes else error

}

}

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Code reusability

2) Template like code

3) Type-safety: We can hold only a single type of objects in generics. It doesn’t allow

storing other objects.

4) Type casting is not required: There is no need to typecast the object. Before Generics, we

need to type cast.

5) Compile-Time Checking: It is checked at compile time so problem will not occur at

runtime. The good programming strategy says it is far better to handle the problem at

compile time than runtime.

6) Only works with reference types i.e. only wrapper classes’ names like Integer, String,

Float etc are passed as a type parameter.

4

Example: Generic class to demo getClass() and getName() methods

class c2<T>

{

T x;

c2(T v)

{x=v;}

void f2()

{ System.out.println(x+" Details are: "+ x.getClass().getName());}

}

class GenP1

{

public static void main(String args[])

{

c2<Integer> ob2=new c2<Integer>(10);

ob2.f2();

c2<String> ob3=new c2<String>("Hello Generics");

ob3.f2();

//works only with reference types i.e. wrapper classes else error

}

}

5

Example: Generic class with two different type parameters (multi parameters)

class c2<T,U>

{

T x;

U y;

c2(T v,U w)

{

x=v;

y=w;

}

void f2()

{

System.out.println(x);

System.out.println(y);

}

}

class GenP2

{

public static void main(String args[])

{

c2<Integer,String> ob2=new c2<Integer,String>(10,"Hello Generics");

ob2.f2();

}

}

6

Example: Program to demo generic method

class c2

{

<T> void f2(T x)

{

System.out.println(x);

}

}

class GenP3

{

public static void main(String args[])

{

c2 ob2=new c2();

ob2.<Integer>f2(10);

}

}