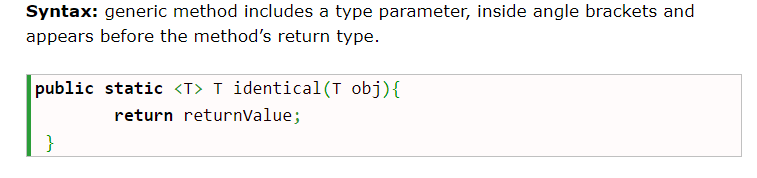
**Generics**

* **Generic** methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
* **Main features of generics:**
* Compile time type safety.
* Removing risk of ClassCastException at run time.
* No casting.

**Generic Methods:**

* **We can create a method that can be used with any type of data. Such a class is known as Generics Method.**
* **The methods which can handle different types of arguments are known as generic methods. Types of the argument are specified on method call.**

****

* Example 1:

class Test

{

//Generics method

public<T> void showItemDetails(T item)

{

System.out.println("Value of the item: " + item);

System.out.println("Type of the item: " + item.getClass().getName());

}

}

class GenericsMethod1

{

public static void main(String args[])

{

//String type test

Test test = new Test();

test.showItemDetails("Test String.");

test.showItemDetails(100);

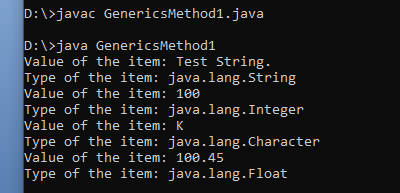
test.showItemDetails('K');

test.showItemDetails(100.45f);

}

}

* **Output**

****

* Example 2: Generics method example with two parameters:

class Test

{

//Generics method with two parameters.

public<T, U> void showItemDetails(T itemT, U itemU)

{

System.out.println("Value of the itemT: " + itemT);

System.out.println("Type of the itemT: " + itemT.getClass().getName());

System.out.println("Value of the itemU: " + itemU);

System.out.println("Type of the itemU: " + itemU.getClass().getName());

}

}

class GenericsMethod2

{

public static void main(String args[])

{

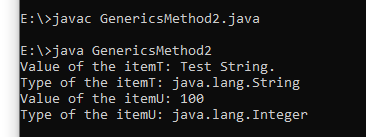
//String type test

Test test = new Test();

test.showItemDetails("Test String.", 100);

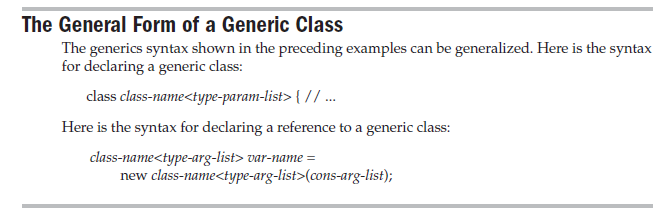
}

}

****

**Generic Classes**

* We can create a class that can be used with any type of data. Such a class is known as Generics Class.



* Example :

class Test<T>

{

private T item;

public Test(T item)

{

this.item = item;

}

public void showItemDetails()

{

System.out.println("Value of the item: " + item);

System.out.println("Type of the item: " + item.getClass().getName());

}

}

Class GenericsTest

{

public static void main(String args[])

{

//String type test

Test<String> test1 = new Test<String>("Test String.");

test1.showItemDetails();

//Integer type test

Test<Integer> test2 = new Test<Integer>(100);

test2.showItemDetails();

//Float type test

Test<Float> test3 = new Test<Float>(100.88f);

test3.showItemDetails();

//Char type test

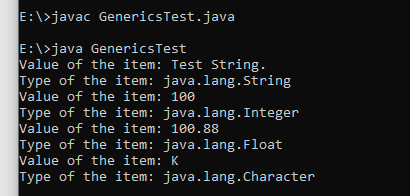
Test<Character> test4 = new Test<Character>('K');

test4.showItemDetails();

}

}

* **Output**



**Generics class example with two parameters:**

class Test<T, U>

{

private T itemT;

private U itemU;

public Test(T itemT, U itemU)

{

this.itemT = itemT;

this.itemU = itemU;

}

public void showItemDetails()

{

System.out.println("Value of the itemT: " + itemT);

System.out.println("Type of the itemT: " + itemT.getClass().getName());

System.out.println("Value of the itemU: " + itemU);

System.out.println("Type of the itemU: " + itemU.getClass().getName());

}

class GenericsTest2

{

public static void main(String args[])

{

//String type test

Test<String, Integer> test = new Test<String, Integer>("Test String.", 100);

test.showItemDetails();

}

}

