Java Reflection is the process of analyzing and modifying all the capabilities of a class at runtime. Reflection API in Java is used to manipulate class and its members which include fields, methods, constructor, etc. at runtime.

One advantage of reflection API in Java is, it can manipulate private members of the class too.

The java.lang.reflect package provides many classes to implement reflection java.Methods of the java.lang.Class class is used to gather the complete metadata of a particular class.

**Class in java.lang.reflect Package**

Following is a list of various Java classes in java.lang.package to implement reflection-

* **Field**: This class is used to gather declarative information such as datatype, access modifier, name and value of a variable.
* **Method**: This class is used to gather declarative information such as access modifier, return type, name, parameter types and exception type of a method.
* **Constructor**: This class is used to gather declarative information such as access modifier, name and parameter types of a constructor.
* **Modifier**: This class is used to gather information about a particular access modifier.

**Methods used in java.lang.Class**

* **Public String getName ()**: Returns the name of the class.
* **public Class getSuperclass()**: Returns the super class reference
* **Public Class[] getInterfaces()** : Returns an array of interfaces implemented by the specified class
* **Public in getModifiers ():** Returns an integer value representing the modifiers of the specified class which needs to be passed as a parameter to "**public static String toString (int i )"** method which returns the access specifier for the given class.

JavaBean

**“A JavaBean is reusable, platform independent component that can be manipulated visually in a builder tool.”**

In computing, based on the Java Platform, JavaBeans are classes that encapsulate many objects into a single object (the bean). Builder tool enables you to create and use beans for application development purpose. In simple words JavaBean is nothing but a Java class. When these JavaBeans are used in other applications, the internal working of such components are hidden from the application developer.

A JavaBean is a Java class that should follow the following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

## ****JavaBeans Properties****

JavaBean property can be access by the user of the object, it can be read, write, read only or write only. We can access these JavaBeans properties with the help of

getPropertyName() method also known as getter or accessor and

setPropertyName() method known as setter written in implementation class of bean.

**Advantages of JavaBeans**

Following are some advantages of JavaBeans:

* Reusability in different environments.
* Used to create applet, servlet, application or other components.
* JavaBeans are dynamic, can be customized.
* Can be deployed in network systems

Simple example of JavaBean clas

**package** mypack;

**public** **class** Employee **implements** java.io.Serializable

{

**private** **int** id;

**private** String name;

**public** Employee(){}

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

**public** **int** getId(){**return** id;}

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

**public** String getName(){**return** name;}

}

 Generic programming enables the programmer to create classes,interfaces and methods in which type of data is specified as a parameter. It provides a facility to write an algorithm independent of any specific type of data. Generics also provide type safety. Type safety means ensuring that an operation is being performed on the right type of data before executing that operation.

Using Generics, it has become possible to create a single class ,interface or method that automatically works with all types of data(Integer, String, Float etc). It has expanded the ability to reuse the code safely and easily.

Before Generics was introduced, generalized classes,interfaces or methods were created using references of type Object

class Gen <T>

{

T ob; //an object of type T is declared<

Gen(T o) //constructor

{

ob = o;

}

public T getOb()

{

return ob;

}

}

class Test

{

public static void main (String[] args)

{

Gen < Integer> iob = new Gen<>(100); //instance of Integer type Gen Class

int x = iob.getOb();

System.out.println(x);

Gen < String> sob = new Gen<>("Hello"); //instance of String type Gen Class

String str = sob.getOb();

System.out.println(str);

}

}

Advantage of Java Generics

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

**1) Type-safety:** We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

Without Generics, we can store any type of objects.

1. List list = **new** ArrayList();
2. list.add(10);
3. list.add("10");
4. With Generics, it is required to specify the type of object we need to store.
5. List<Integer> list = **new** ArrayList<Integer>();
6. list.add(10);
7. list.add("10");// compile-time error

**2) Type casting is not required:** There is no need to typecast the object.

Before Generics, we need to type cast.

1. List list = **new** ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting
4. After Generics, we don't need to typecast the object.
5. List<String> list = **new** ArrayList<String>();
6. list.add("hello");
7. String s = list.get(0);

**3) 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.

1. List<String> list = **new** ArrayList<String>();
2. list.add("hello");
3. list.add(32);//Compile Time Error

**Syntax** to use generic collection

1. ClassOrInterface<Type>

**Example** to use Generics in java

1. ArrayList<String>

# Bounded Type Parameters

There may be times when you want to restrict the types that can be used as type arguments in a parameterized type. To declare a bounded type parameter, list the type parameter's name, followed by the extends keyword, followed by its *upper bound*, which in this example is Number. 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.

Note that, in this context, extends is used in a general sense to mean either "extends" (as in classes) or "implements" (as in interfaces).

public class Box<T> {

private T t;

public void set(T t) {

this.t = t;

}

public T get() {

return t;

}

public <U **extends Number**> void inspect(U u){

System.out.println("T: " + t.getClass().getName());

System.out.println("U: " + u.getClass().getName());

}

public static void main(String[] args) {

Box<Integer> integerBox = new Box<Integer>();

integerBox.set(new Integer(10));

integerBox.inspect("some text"); // **error: this is still String!**

}

}

**Unbounded Wildcards**

The unbounded wildcard type is specified using the wildcard character (?), for example, List<?>. This is called a *list of unknown type*. There are two scenarios where an unbounded wildcard is a useful approach:

* If you are writing a method that can be implemented using functionality provided in the Object class.
* When the code is using methods in the generic class that don't depend on the type parameter. For example, List.size or List.clear. In fact, Class<?> is so often used because most of the methods in Class<T> do not depend on T.

public static void printList(List<?> list) {

for (Object elem: list)

System.out.print(elem + " ");

System.out.println();

}

Because for any concrete type A, List<A> is a subtype of List<?>, you can use printList to print a list of any type:

List<Integer> li = Arrays.asList(1, 2, 3);

List<String> ls = Arrays.asList("one", "two", "three");

printList(li);

printList(ls);

Type Erasure in Java   
Generics concept is introduced in Java language to provide tighter type checks at compile time and to support generic programming. The way to implement generics, the Java compiler applies **type erasure** to:

* Replace all type parameters in generic types with their bounds or Object if the type parameters are unbounded. The produced bytecode, therefore, contains only ordinary classes, interfaces, and methods.
* Insert type casts if necessary to preserve type safety.
* Generate bridge methods to preserve [polymorphism](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/) in extended generic types.

|  |
| --- |
| // Here, T is bounded by Object i.e. java.lang.Object  class GFG<T> {  // Here, T will be replaced by default i.e. Object      T obj;        GFG(T o)      {          obj = o;      }      T getob()      {          return obj;      }  } |

**After compilation, the code is replaced by default Object like the below:**

class GFG

{

// Here, T will be replaced by default i.e. Object

Object obj;

GFG(Object o)

{

obj=o;

}

Object getob()

{

return obj;

}

}

|  |
| --- |
| // Here, T is bounded by Object i.e. java.lang.Object  class Geeks<T extends String> {     // Here, T will be replaced by String i.e. java.lang.String      T str;      Geeks(T o)      {          str = o;      }      T getob()      {          return str;      }  } |

**After compilation, the code is replaced by String like the below:**

class Geeks

{

//Here, T will be replaced by String i.e. java.lang.String

String str;

Geeks(String o)

{

str=o;

}

String getob()

{

return str;

}

}

# DriverManager class

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

# Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

# Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

# ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

### **Commonly used methods of ResultSet interface**

|  |  |
| --- | --- |
| **1) public boolean next():** | is used to move the cursor to the one row next from the current position. |
| **2) public boolean previous():** | is used to move the cursor to the one row previous from the current position. |
| **3) public boolean first():** | is used to move the cursor to the first row in result set object. |
| **4) public boolean last():** | is used to move the cursor to the last row in result set object. |
| **5) public boolean absolute(int row):** | is used to move the cursor to the specified row number in the ResultSet object. |
| **6) public boolean relative(int row):** | is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative. |
| **7) public int getInt(int columnIndex):** | is used to return the data of specified column index of the current row as int. |
| **8) public int getInt(String columnName):** | is used to return the data of specified column name of the current row as int. |
| **9) public String getString(int columnIndex):** | is used to return the data of specified column index of the current row as String. |
| **10) public String getString(String columnName):** | is used to return the data of specified column name of the current row as String. |