**STREAM**

**https://www.geeksforgeeks.org/stream-in-java/**

**Intermediate Operations:**

1. **map:**The map method is used to map the items in the collection to other objects according to the Predicate passed as argument.  
   List number = Arrays.asList(2,3,4,5);  
   List square = number.stream().map(x->x\*x).collect(Collectors.toList());
2. **filter:** The filter method is used to select elements as per the Predicate passed as argument.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
3. **sorted:** The sorted method is used to sort the stream.  
   List names = Arrays.asList("Reflection","Collection","Stream");  
   List result = names.stream().sorted().collect(Collectors.toList());

**Terminal Operations:**

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.  
   List number = Arrays.asList(2,3,4,5,3);  
   Set square = number.stream().map(x->x\*x).collect(Collectors.toSet());
2. **forEach:** The forEach method is used to iterate through every element of the stream.  
   List number = Arrays.asList(2,3,4,5);  
   number.stream().map(x->x\*x).forEach(y->System.out.println(y));
3. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.  
   The reduce method takes a BinaryOperator as a parameter.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

|  |
| --- |
| //a simple program to demonstrate the use of stream in java  **import** java.util.\*;  **import** java.util.stream.\*;    **class** Demo  {  **public** **static** **void** main(String args[])    {        // create a list of integers      List<Integer> number = Arrays.asList(2,3,4,5);        // demonstration of map method      List<Integer> square = number.stream().map(x -> x\*x).                             collect(Collectors.toList());      System.out.println(square);        // create a list of String      List<String> names =                  Arrays.asList("Reflection","Collection","Stream");        // demonstration of filter method      List<String> result = names.stream().filter(s->s.startsWith("S")).                            collect(Collectors.toList());      System.out.println(result);        // demonstration of sorted method      List<String> show =              names.stream().sorted().collect(Collectors.toList());      System.out.println(show);        // create a list of integers      List<Integer> numbers = Arrays.asList(2,3,4,5,2);        // collect method returns a set      Set<Integer> squareSet =           numbers.stream().map(x->x\*x).collect(Collectors.toSet());      System.out.println(squareSet);        // demonstration of forEach method      number.stream().map(x->x\*x).forEach(y->System.out.println(y));        // demonstration of reduce method  **int** even =         number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);        System.out.println(even);    }  } |

//remove item from a list  
  
List<String> fruits = new ArrayList<>();  
fruits.add("Apple");  
fruits.add("Banana");  
fruits.add("Stawberry");  
  
Iterator<String> fruitIterator = fruits.iterator();  
  
while(fruitIterator.hasNext()) {  
 if(fruitIterator.next().contains("Apple")) {  
 fruitIterator.remove();  
 }  
}

System.*out*.println (fruits);

// defference between List - Set  
  
List<String> data = Arrays.*asList*("ab","fc","cd","ab","bc","cd");  
  
List<String> list = new ArrayList<>(data);// will add data as is  
Set<String> set1 = new HashSet<>(data); // will add data keeping only unique values  
Set<String> set2 = new TreeSet<>(data); // will add data keeping unique values and sorting  
Set<String> set3 = new LinkedHashSet<>(data); // will add data keepig only unique values and preserving the original order  
  
System.*out*.println(list);  
System.*out*.println(set1);  
System.*out*.println(set2);  
System.*out*.println(set3);

**UnModiFiableList -->ReadOnly**

public class UnModifiablList {  
  
 private List<Integer> intList = new ArrayList<>();  
  
 public void addValue(Integer value) {  
 intList.add(value);  
 }  
  
 public List<Integer> getIntList() {  
 return Collections.*unmodifiableList*(intList);

**Check Funtional Interface**

package Main;  
  
@FunctionalInterface  
public interface bar {  
 void barInt();  
 default int batDouble() { return 0};  
}

**GENERIC CLASS**

Unlike other identifiers, type parameters have no naming constraints. However their names are commonly the first  
letter of their purpose in upper case. (This is true even throughout the official JavaDocs.)  
Examples include T for "type", E for "element" and K/V for "key"/"value"

public class TestGeneric<T> {  
  
 private T value;  
  
 public void setValue(T value) {  
 this.value=value;  
 }  
  
 public T getValue(){  
 return this.value;  
 }  
}

**DELETE FILE**

File file = new File("d:\\InputData\\JavaTakeNote.docx");  
if(file.exists()) {  
 boolean deleteStatus = file.delete();  
 if(!deleteStatus) {   
 throw new IOException("Unable to delete file");  
 }  
} else {  
 throw new IOException("File does not exist !" );  
}

**Iterating on each file within a folder**

File file = new File("d:\\InputData");  
for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );  
  
}

**RECURSE FOLDER**

public static void recurseFolder (String path) {  
 File file = new File(path);  
 for(File seletedFile : file.listFiles()) {  
 String property = seletedFile.isFile() ? "file" :"directory";  
 if(property.equals("directory")){  
 *recurseFolder*(seletedFile.getAbsolutePath());  
 }  
  
 System.*out*.println (property + " " + seletedFile.getAbsolutePath() );

}  
}

**COPY FILE**

public static void copyFile(String sourceFile, String desFile) {  
 File source = new File (sourceFile);  
 File des = new File(desFile);  
 try {  
 FileChannel sourceChanel = new FileInputStream(source).getChannel();  
 FileChannel desChanel = new FileOutputStream(des).getChannel();  
 sourceChanel.transferTo(0,sourceChanel.size(),desChanel);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

*// Read from baseReader, one line at a time*BufferedReader reader = new BufferedReader( baseReader );  
String line;  
while((line = reader.readLine()) != null) {  
*// Remember: System.out is a stream, not a writer!*System.out.println(line);  
}

**READ FILE USE SCANNER**

public static void readFile(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNextLine()) {  
 System.*out*.println(sc.nextLine());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

public static void readWoedByWord(String sourceFile) {  
 try (Scanner sc = new Scanner(new File(sourceFile))) {  
  
 while (sc.hasNext()) {  
 System.*out*.println(sc.next());  
 }  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
}

**CREATE A NEW FOLDER**

public static void makeFolder() {  
 String path ="d:\\InputData\\NewFolder";  
 File file = new File (path);  
 file.mkdir();  
  
}

**WRITE TEXT FILE**

public static void writeLineToFiile() {  
 String pathFile = "D:\\log.txt";  
 String line ="Level log : fine";  
 try {  
 BufferedWriter wr = new BufferedWriter(new FileWriter(pathFile));  
 wr.write(line);  
 wr.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

**READ TEXT FILE**

public static void readFile() {  
  
 String pathFile = "d:\\InputData\\strorage.xml";  
 try {  
 BufferedReader br = new BufferedReader(new FileReader(pathFile));  
 String s =null;  
 while ((s=br.readLine())!=null){  
 System.*out*.println(s);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**SINGLETON PATTERN USE ENUM**

package Main;  
  
public enum SingleTon {  
  
 *INSTANCE*;  
  
 private SingleTon () {}  
  
 public void f() {  
 System.*out*.println("CODE HERE");  
  
 }  
}

**READ TEXT DATA FROM WEB**

public static void readWeb() {  
  
 String url = "https://vnexpress.net/";  
 try {  
 HttpsURLConnection httpCon = (HttpsURLConnection)new URL(url).openConnection();  
 int responsecode = httpCon.getResponseCode();  
 if(responsecode>=200&&responsecode<=299) {  
 InputStream inputstream = httpCon.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputstream));  
 String line;  
  
 while ((line =br.readLine())!=null) {  
 System.*out*.println(line);  
 }  
  
 br.close();  
  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**DATE FORMAT**

public static void printDateFormat() {  
  
 String format ="yyyy/MM/dd hh:mm:ssXXX";  
  
 Date date = Calendar.*getInstance*().getTime();  
  
 SimpleDateFormat simpleDateFormat = new SimpleDateFormat(format);  
  
 String formatedDate = simpleDateFormat.format(date);  
  
 System.*out*.println(date);  
 System.*out*.println(formatedDate);  
  
  
}

**REFLECTION**

public static void reflection() {  
  
 Class<String> clazz = String.class;  
 Constructor<?>[] constructor = clazz.getConstructors();  
 Method[]methods = clazz.getMethods();  
  
 for(Method method : methods) {  
 System.*out*.println(method);  
 }  
  
 for(Constructor c : constructor) {  
 System.*out*.println(c);  
 }  
  
}

**INVOKE FUNCTION USE REFLECTION**

public static void reflection() {  
  
 Class<String> clazz = String.class;  
 Constructor<?>[] constructor = clazz.getConstructors();  
 try {  
 Method method= clazz.getMethod("length");  
  
 System.*out*.println(method.invoke("Hello Reflection"));  
  
 } catch (NoSuchMethodException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InvocationTargetException e) {  
 e.printStackTrace();  
 }

**CREATE RANDOM NUMBER**

public static void createRandomNumber() {  
  
 SecureRandom rng = new SecureRandom();  
 System.*out*.println(rng.nextInt());  
  
}

**PROCESS EXCUTE**

public static void process() {  
 try {  
 Process p = Runtime.*getRuntime*().exec("ipconfig");  
 InputStream inputstream = p.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(inputstream));  
 String line;  
 while ((line=br.readLine())!=null) {  
 System.*out*.println(line);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
}

**PARSE XML**

public static Document parseXMLDocument(String fileName) {  
  
 File xmlFile = new File(fileName);  
 if(!xmlFile.exists()) {  
 System.*out*.println("File " + fileName + " does not exist");  
 }  
 try {  
 DocumentBuilder db = DocumentBuilderFactory.*newInstance*().newDocumentBuilder();  
 return db.parse(fileName);  
 } catch (ParserConfigurationException e) {  
 e.printStackTrace();  
 } catch (SAXException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
  
}

public void temp() {  
 String pathVC = System.*getProperty*("user.dir")+"\\datamodel\\vendorcerts\\test\\";  
  
 File files = new File(pathVC);  
 File []listVC = files.listFiles();  
 for(File vc : listVC) {  
 Document dom = XMLUtil.*parseXMLDocument*(vc.getAbsolutePath());  
 NodeList nListFacetType= dom.getElementsByTagName("FacetType");  
 Node nFacetType = nListFacetType.item(0);  
 Element eFacetType = (Element)nFacetType;  
 System.*out*.println(eFacetType.getAttribute("name")); }  
  
 System.*out*.println("Done");  
}

**GET ENVIRONMENT VARIABLEs**

public static void main(String []arg) throws IOException {  
  
 System.*out*.println(System.*getenv*());

System.*out*.println(System.*getenv*("PATH"));  
  
}

**GET PROPERTY INFO**

public static void main(String []arg) throws IOException {  
  
 System.*out*.println(System.*getProperties*());  
 System.*out*.println(System.*getProperty*("user.dir"));  
  
}

**READ INPUT STREAM OF A FILE**

public static void process() {  
  
 String input = "d:\\InputData\\AciStorage.json";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 byte[] buf = new byte[1024];  
 int temp = 0;  
 while((temp=bis.read(buf))!=-1) {  
 System.*out*.println(temp);  
 System.*out*.println(buf[2]);  
 }  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**COVERT A FILE CONTENT TO BYTE**

public static void process() {  
  
 String input = "d:\\InputData\\AciStorage.json";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 byte[] buf = new byte[1024];  
 int len = 0;  
 while((len=bis.read(buf))!=-1) {  
 baos.write(buf, 0, len);  
 }  
 baos.flush();  
  
 byte [] result = baos.toByteArray();  
   
 System.*out*.println(result[2]);  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

public static void process() {  
  
 String input = "d:\\InputData\\temp.txt";  
 FileInputStream fileInputStream = null;  
 try {  
 fileInputStream = new FileInputStream(input);  
 BufferedInputStream bis = new BufferedInputStream(fileInputStream);  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 byte[] buf = new byte[1024];  
 int len = 0;  
 while((len=bis.read(buf))!=-1) {  
 baos.write(buf, 0, len);  
 }  
 baos.flush();  
  
 byte [] result = baos.toByteArray();  
  
 String s = new String(result);  
  
 System.*out*.println(s.length());  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
}

**TEST JMS**

**Pom.XML**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>groupId</groupId>  
 <artifactId>JavaTest</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <dependency>  
 <groupId>org.wso2.sp</groupId>  
 <artifactId>jms-producer</artifactId>  
 <version>4.2.0-M2</version>  
 <type>pom</type>  
 </dependency>  
 <dependency>  
 <groupId>javax.jms</groupId>  
 <artifactId>javax.jms-api</artifactId>  
 <version>2.0</version>  
 </dependency>  
 </dependencies>  
</project>

**CONNECT TO DATABASE**

Driver: Knows how to get a connection to the database  
Connection: Knows how to communicate with the database  
Statement: Knows how to run the SQL  
ResultSet: Knows what was returned by a SELECT query

1. **Config Setting.xml (download driver)**

**Add :**

**<repository>**

**<id>my-repo2</id>**

**<name>sql-lib</name>**

**<url>http://clojars.org/repo/</url>**

**</repository>**

1. **Code**

import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 Statement st = null;  
  
 try {  
 System.*out*.println("Begin");  
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
 conn = DriverManager.*getConnection*("jdbc:sqlserver://localhost:1433", "sa", "123456");  
 st = conn.createStatement();  
 String comand = "CREATE DATABASE VendorCertInfo";  
 st.executeUpdate(comand);  
 System.*out*.println("Done");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }

**GET DATA FROM DATABASE**

import java.sql.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 Statement st = null;  
 ResultSet result =null;  
  
 try {  
 System.*out*.println("Begin");  
   
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
  
 String url= "jdbc:sqlserver://localhost:1433;databaseName=CertProject;user=sa;password=123456";  
  
 conn = DriverManager.*getConnection*(url);  
  
 st = conn.createStatement();  
  
 String comand = "SELECT \*FROM ColumExpression";  
  
 result = st.executeQuery(comand);  
  
 while (result.next()) {  
 String value = result.getString("Expression");  
 System.*out*.println(value);  
 }  
  
 System.*out*.println("Done");  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }

import java.sql.\*;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Connection conn = null;  
 PreparedStatement st = null;  
 ResultSet result =null;  
  
 try {  
 System.*out*.println("Begin");  
  
 Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");  
  
 String url= "jdbc:sqlserver://localhost:1433;databaseName=CertProject;user=sa;password=123456";  
  
 conn = DriverManager.*getConnection*(url);  
  
  
  
 String comand = "SELECT \*FROM ColumExpression";  
  
 st = conn.prepareStatement(comand);  
 result = st.executeQuery();  
  
 while (result.next()) {  
 String value = result.getString("Expression");  
 System.*out*.println(value);  
 }  
  
 System.*out*.println("Done");  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
}

**Lambda Expression (OCA\_ Oracle Certified Associate Java SE 8 Programmer I Study Guide\_ Exam 1Z0-808-Sybex (2014).pdf)**

public class Animal {  
  
 private String species;  
 private boolean canHop;  
 private boolean canSwim;  
 public Animal(String speciesName, boolean hopper, boolean swimmer) {  
 species = speciesName;  
 canHop = hopper;  
 canSwim = swimmer;  
 }  
 public boolean canHop() {  
 return canHop;  
 }  
  
 public boolean canSwim() {  
 return canSwim;  
 }  
  
 public String toString() {  
 return species;  
 }  
  
}

public interface CheckTrait {  
 boolean test(Animal a);  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class Test {  
  
 public static void main(String []arg) {  
 List<Animal> animals = new ArrayList<Animal>(); // list of animals  
 animals.add(new Animal("fish", false, true));  
 animals.add(new Animal("kangaroo", true, false));  
 animals.add(new Animal("rabbit", true, false));  
 animals.add(new Animal("turtle", false, true));  
  
 *print*(animals,a->a.canSwim());  
  
 }  
  
 private static void print(List<Animal> animals, CheckTrait checker) {  
  
 for (Animal animal : animals) {  
 if (checker.test(animal))  
 System.*out*.print(animal + " ");  
 }  
 System.*out*.println();  
 }  
  
}

public class Animal {  
  
  
 public boolean canHop() {  
 System.*out*.println("This is can Hop");  
 return true;  
 }  
  
 public boolean canSwim() {  
 System.*out*.println("This is can Swim");  
 return false;  
 }  
  
  
  
}

public interface CheckTrait {  
 boolean test();  
}

import java.util.ArrayList;  
import java.util.List;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 Animal a = new Animal();  
  
 *print*(()->a.canHop());  
  
 }  
  
 private static void print( CheckTrait checker) {  
  
 System.*out*.println (checker.test());  
  
 }  
  
}

**THREAD**

public class CreateThread {  
  
 public void thread1 () {  
 for(int i =0; i < 20;++i) {  
 System.*out*.println(Thread.*currentThread*().getName() + " 111 " + i);  
 }  
 }  
  
 public void thread2 () {  
 for(int i =0; i < 10;++i) {  
 System.*out*.println(Thread.*currentThread*().getName() + " 222 " + i);  
 }  
 }  
  
  
}

public class Test {  
  
 public static void main(String []arg) {  
  
 CreateThread ct = new CreateThread();  
 *startThread*(()->ct.thread1());  
 *startThread*(()->ct.thread2());  
  
 }  
  
 public static void startThread(Runnable r) {  
  
 Thread t = new Thread(r);  
 t.start();  
  
 }  
  
  
}

# **Annotations**

Creating an annotation requires two pieces of information:

1. a retention policy and (2) a target

A ****retention policy**** specifies how long, in terms of the program lifecycle, the annotation should be retained for

The ****target**** of an annotation specifies which Java constructs an annotation can be applied to

|  |  |
| --- | --- |
| **Element Types** | **Where the annotation can be applied** |
| TYPE | class, interface or enumeration |
| FIELD | fields |
| METHOD | methods |
| CONSTRUCTOR | constructors |
| LOCAL\_VARIABLE | local variables |
| ANNOTATION\_TYPE | annotation type |
| PARAMETER | parameter |

import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*FIELD*)  
public @interface JsonField {  
 public String value() default "";  
}

public class Car {  
  
 @JsonField("Model")  
 private final String model;  
  
 @JsonField("Manufacture")  
 private final String make;  
  
  
 public Car(String make, String model) {  
 this.make = make;  
 this.model = model;  
 }  
 public String getMake() {  
 return make;  
 }  
 public String getModel() {  
 return model;  
 }  
}

import java.lang.reflect.Field;  
import java.lang.reflect.Method;  
import java.util.HashMap;  
import java.util.Map;  
  
import static java.util.Objects.*requireNonNull*;  
import static java.util.stream.Collectors.*joining*;  
  
public class JsonSerializer {  
  
 public String serialize(Object object) {  
  
 Class<?> objectClass = *requireNonNull*(object).getClass();  
  
 for(Method method : objectClass.getMethods()) {  
 System.*out*.println("Method : " + method.getName());  
 }  
  
 Map<String, String> jsonElements = new HashMap<>();  
  
 for (Field field: objectClass.getDeclaredFields()) {  
  
 System.*out*.println("Field value : "+ field.getName());  
  
 field.setAccessible(true);  
  
 if (field.isAnnotationPresent(JsonField.class)) {  
  
 try {  
 jsonElements.put(*getSerializedKey*(field), (String) field.get(object));  
 //System.out.println("jsonElement : " + jsonElements);  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
 }  
 return toJsonString(jsonElements);  
 }  
  
 private String toJsonString(Map<String, String> jsonMap) {  
 String elementsString = jsonMap.entrySet()  
 .stream()  
 .map(entry -> "\"" + entry.getKey() + "\":\"" + entry.getValue() + "\"")  
 .collect(*joining*(","));  
 return "{" + elementsString + "}";  
 }  
  
 private static String getSerializedKey(Field field) {  
 String annotationValue = field.getAnnotation(JsonField.class).value();  
 if (annotationValue.isEmpty()) {  
 return field.getName();  
 }  
 else {  
 return annotationValue;  
 }  
 }  
}

**REFLECTION + RESOURCEHEADER**

import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*TYPE*)  
public @interface ResourceHeader {  
 public String value() default "";  
}

@ResourceHeader("controller")  
public class Car {  
  
  
 public void custom() {  
  
 System.*out*.println("custom of Car class is calling");  
  
 }  
}

import java.lang.annotation.Annotation;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
  
import static java.util.Objects.*requireNonNull*;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
 String value ="controller";  
  
 Car car = new Car();  
  
 Class<?> object = car.getClass();  
 Annotation[] ans = object.getAnnotations();  
  
 for(Annotation a : ans) {  
 if(a instanceof ResourceHeader){  
 ResourceHeader r = (ResourceHeader)a;  
 String file = r.value();  
  
 if(file.equals(value)) {  
  
 try {  
 Method method = object.getMethod("custom");  
 method.invoke(car);  
 } catch (NoSuchMethodException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
 }  
  
  
  
  
  
 }  
  
  
  
}

**SERIALIZING OBJECTS**

**The process of storing and retrieving objects in an external fi le is called *serialization.***

**WRITING AN OBJECT TO A FILE**

import java.io.ObjectOutputStream;  
import java.io.OutputStream;  
import java.io.Serializable;  
import java.nio.file.Files;  
  
public class Car implements Serializable {  
  
 private static final long *serialVersionUID* = 9002L;  
 private int model;  
  
 public void custom() {  
  
 System.*out*.println("custom of Car class is calling");  
  
  
 }  
}

public class Test {  
  
 public static void main(String []arg) {  
  
 Car c = new Car();  
  
 Path file = Paths.*get*("d:\\DataFile\\Object.txt");  
  
 try {  
 ObjectOutputStream out = new ObjectOutputStream(Files.*newOutputStream*(file));  
  
 out.writeObject(c);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }

**READING AN OBJECT FROM A FILE**

import java.io.BufferedInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
  
import static java.util.Objects.*requireNonNull*;  
  
public class Test {  
  
 public static void main(String []arg) {  
  
  
  
 Path file = Paths.*get*("d:\\DataFile\\Object.txt");  
  
 try {  
 ObjectInputStream in = new ObjectInputStream(new BufferedInputStream(  
 Files.*newInputStream*(file)));  
  
 Car obj = (Car)in.readObject();  
 obj.custom();  
  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

**ZIP FOLDER**

import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.zip.ZipEntry;  
import java.util.zip.ZipOutputStream;  
  
public class ZipFolder {  
  
  
 List<String> listFile = new ArrayList<>();  
  
 public void zipFolder(File dir, String zipFolder) {  
 List<String> files = listAllFile(dir);  
 try {  
  
 FileOutputStream fos = new FileOutputStream(zipFolder);  
 ZipOutputStream zos = new ZipOutputStream(fos);  
 for(String filePath : files) {  
 String relativePath = filePath.substring(dir.getAbsolutePath().length()+1);  
 ZipEntry ze = new ZipEntry(relativePath);  
 FileInputStream fis = new FileInputStream(filePath);  
 zos.putNextEntry(ze);  
 byte[] buffer = new byte[1024];  
 int len = 0;  
  
 while((len = fis.read(buffer))>0){  
 zos.write(buffer,0,len);  
 }  
 zos.closeEntry();  
 fis.close();  
  
 }  
  
 zos.close();  
 fos.close();  
  
  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
  
  
 public List<String> listAllFile(File dir) {  
  
 File[] files = dir.listFiles();  
  
  
 for(File f : files ) {  
 if(f.isFile()) {  
 listFile.add(f.getAbsolutePath());  
  
 } else{  
 listAllFile(f);  
 }  
 }  
  
 return listFile;  
 }  
}