Difference between ArrayList and Vector:

	ArrayList		Vector
1	It is not synchronized, multiple threads can work on ArrayList at the same time.	1	It is synchronized, which means only one thread can access the code at a time.
2	ArrayList increases half of its size when its size is increased.	2	Vector doubles size of array when its size is increased.
3	ArrayList is fast because it is non-synchronized	3	Vector is slow because it is synchronized.
4	ArrayList uses the Iterator interface to traverse the elements.	4	A Vector can use the Iterator interface or Enumeration interface to traverse the elements.

Stack:

In Java Stack class is a child class of Vector class in collection framework. The Stack class is based on the principle of Last in First out (LIFO). The element inserted last will be the first to be removed.

Stack Methods:

push(): Places an element on the stack.

pop(): Takes the top element from the stack.

peek(): Returns the top element of the stack without removing it from the stack.

empty(): It returns true if the stack is empty, otherwise returns false.

Example of Stack:

```
import java.util.Stack;
public class StackDemo {
	public static void main(String[] args) {
		Stack<Integer> stack = new Stack<Integer>();
		stack.push(25);
		stack.push(70);
		stack.push(35);
		stack.push(40);
		stack.push(45);
		stack.push(50);

		System.out.println("Elements of Stack: " +stack);
		System.out.println("Element at the top of stack: " +stack.peek());
		System.out.println("Position of element 50: " +stack.search(50));
		System.out.println();

		while(stack.size() > 0){
			System.out.println("Removed element: " +stack.pop());
		 }
		System.out.println("Is stack empty: " +stack.empty());
	}
}
```

Output:

```
Elements of Stack: [25, 70, 35, 40, 45, 50]
Element at the top of stack: 50
Position of element 50: 1

Removed element: 50
Removed element: 45
Removed element: 40
Removed element: 35
Removed element: 70
Removed element: 25
Is stack empty: true
```

HashTable:
HashTable is a collection class. It maps keys to values. Similar to HashMap, it also stores the data in key, value pair. HashTable doesn't allow null for boths key and value.

Example of HashTable:

```
import java.util.Enumeration;
import java.util.Hashtable;

public class HashTableDemo {
        public static void main(String[] args) {

                Hashtable hashtable = new Hashtable();
                hashtable.put(1, "Abhiraj");
                hashtable.put(2, "Rahit");
                hashtable.put("A", "Kabi");
                System.out.println(hashtable);
                System.out.println();

                Hashtable<Integer,String> hashtable1=new Hashtable<Integer,String>();
                hashtable1.put(new Integer(2), "Two");
                hashtable1.put(new Integer(1), "One");
                hashtable1.put(3, "Three");
                System.out.println(hashtable1);
                System.out.println();

                Enumeration em=hashtable1.keys();

    while(em.hasMoreElements())
    {
      //nextElement is used to get key of Hashtable
      int key = (Integer)em.nextElement();

      //get is used to get value of key in Hashtable
      String value=(String)hashtable1.get(key);

      System.out.println("Key :"+key+"  value :"+value);}}}
```

Output:

```
{A=Kabi, 2=Rahit, 1=Abhiraj}

{3=Three, 2=Two, 1=One}

Key :3  value :Three
Key :2  value :Two
Key :1  value :One
```

Difference between HashMap and HashTable:

HashMap		HashTable	
1	HashMap allows null value for both key and value.	1	HashTable doesn't allows null value for both key and value.
2	It is unsynchronized.	2	It is synchronized.
3	HashMap is fast.	3	Hashtable is slow.

HashMap:

HashMap is a collection class based on Map. We use it to store Key & value pairs. You denote a HashMap Java as HashMap<K, V>, where K stands for Key and V stands for Value.

Example of HashMap:

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapDemo {
	public static void main(String[] args) {
		Map<Integer, String> hashMap = new HashMap<Integer, String>();

		hashMap.isEmpty();
		System.out.println(hashMap);

		hashMap.put(1, "Alka");
		hashMap.put(2, "Rimi");
		hashMap.put(null, null);
		hashMap.put(4, "Shaan");
		hashMap.put(4, "Abhiraj");

		System.out.println(hashMap);

		System.out.println(hashMap.size());

		hashMap.remove(4);

		System.out.println(hashMap);

		Set set = hashMap.entrySet();

	Iterator it = set.iterator();

	while(it.hasNext())
	{
	 Map.Entry me = (Map.Entry)it.next();

	System.out.println("This is Key = "+me.getKey() + " : "+"This is Values = "+ me.getValue() );
	}
	System.out.println(hashMap.entrySet());
	}

}
```

Output:

```
{}
{null=null, 1=Alka, 2=Rimi, 4=Abhiraj}
4
{null=null, 1=Alka, 2=Rimi}
This is Key = null : This is Values = null
This is Key = 1 : This is Values = Alka
This is Key = 2 : This is Values = Rimi
[null=null, 1=Alka, 2=Rimi]
```

Exception Handling:
Exception is a run-time error. So we need the exception handling because during the run time our code can generates any error and that will cause to terminates the application or program, so due to that reason we are using exception handling to handle without exception without interrupting the our program.

Type of Exceptions:
There are two types of exception.

1. Checked exception
2. Unchecked exception

Checked exception:
The checked exception is an exception that is checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If it is not handled, the compiler displays a compilation error using built-in classes.

There are a few built-in classes used to handle checked exceptions.

- IOException
- FileNotFoundException
- ClassNotFoundException
- SQLException
- DataAccessException
- InstantiationException
- UnknownHostException

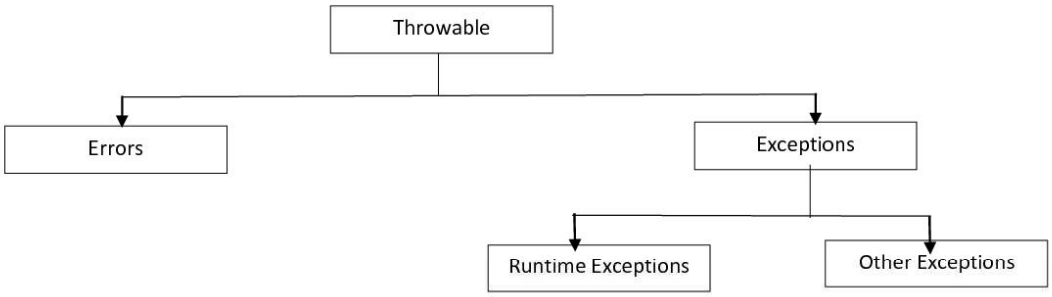
Unchecked exception:
The unchecked exception is an exception that occurs at the time of program execution. The unchecked exceptions are not caught by the compiler at the time of compilation.

There are a few built-in classes used to handle unchecked exceptions.

- **ArithmeticException**
- **NullPointerException**
- **NumberFormatException**
- **ArrayIndexOutOfBoundsException**
- **StringIndexOutOfBoundsException**

Advantage of exception handling:
Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid-way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly. By handling we make sure that all the statements execute and the flow of program doesn't break.

Exception Hierarchy:
Throwable class is the Super or root class of Java Exception hierarchy which contains two subclasses: Exception and Error.



Example of exception:

```
public class ExceptionThrowError {
        public static void main(String[] args) {

                int a=10;
                int b=0;
                int c=a/b;
                System.out.println(c);
        }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at
com.sun.exceptions.ExceptionThrowError.main(ExceptionThrowError.java:13)
```

How to Handle Exception:
The exception handling mechanism uses five keywords namely

1. try
2. catch
3. finally
4. throw
5. throws

try: The try block contains a set of statements where an exception can occur.

catch: The main purpose of catch block is to handle the exception which are throws by the try block. If catch block will not be executed if there is no exception inside the try block.

Syntax of try and catch block:

```
try {
        // Block of code to try
}
catch(ExceptionType1 e) {
        // Block of code to handle errors
}
```

finally:
The finally keyword used to define a block that must be executed irrespective of exception occurrence.

Syntax of finally block:

```
try {
                        // Block of code to try

}
catch (ExceptionType1 e1) {
                        // Block of code to handle errors }
finally {
                        // finally block always executes
}
```

throw:
If a user wants to throw same specific error which is easy to understand by user then we use our own exception with help of throw block.

Syntax of throw block:

```
throw new exception_class("error message")
```

throws: The throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

Syntax of throws block:

```
type method_name(parameter_list) throws exception_list
{
 // definition of method }
```

Difference between throw and throws:

	throw		throws
1	Throw keyword always present inside the method body.	1	Throws keyword always used with the method signature.
2	We can throw only one exception at a time	2	We can handle multiple exception using throws keyword
3	Throw keyword is used to throw an exception object explicitly	3	Throws keyword is to declare an exception as well as by pass the method
4	Throw is followed by an instance	4	Throws is followed by class

Example of try catch block:

```
public class ExceptionTryCathBlock {
	public static void main(String[] args) throws Exception {
		int a = 10;
		int b = 2;
		int c;

		try {
			c = a / b;
			System.out.println("This will not be printed.");
		}
		catch (ArithmeticException e) {
			System.out.println("Division by zero.");
		}
		finally {
			System.out.println("This is Finally block");
		}
	}
}
```

Example of throw and throws block:

```
import java.util.Scanner;

public class ExceptionThrowThrowsBlock {

	public void readAge(int age) throws Exception {
		if (age < 15 || age > 60)

			throw new Exception();

		else
			System.out.println("Your Age is : " + age);
	}

	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		System.out.print("Please Enter Your Age : ");
		int age = in.nextInt();

		ExceptionThrowThrowsBlock obj = new ExceptionThrowThrowsBlock();
		try {
			obj.readAge(age);
		} catch (Exception e) {

		System.out.println(" Age Must be Betwee 15 and 60 ..Age Entered is "+ age);
		}
		finally {
			System.out.println("This is Finally block");
		}
	}

}
```

Output:

```
Please Enter Your Age : 100
Age Must be Betwee 15 and 60 ..Age Entered is 100
This is Finally block
```

```
Please Enter Your Age : 50
Your Age is : 50
This is Finally block
```

Example of multiple try block and multiple catch block:

```
public class ExceptionMultipleTryBlockAndCatch {
	public static void main(String[] args) {
		try {
			int a=5;
			int b=0;
			int div;
			div=a/b;
			System.out.println("This will not be printed."+div);
		} catch (ArithmeticException a) {

			System.out.println("Division by zero.");
			System.out.println(a.getMessage());
		}
		try {
			int arr[]={10,20,30,40};
			System.out.println(arr[5]);

		} catch (ArrayIndexOutOfBoundsException o) {
			System.out.println("Array out of index");
			System.out.println(o.getMessage());
		}}}
```

Output:

```
Division by zero.
/ by zero
Array out of index
5
```

Example of multiple catch blocks:

```
public class ExceptionMultiplCatchBlock {

	public static void main(String[] args) throws Exception {
		try {
			int a = 10, b = 5, div;
			div = a / b;
			System.out.println(div);

			int arr[] = { 10, 20, 30, 40 };
			System.out.println(arr[5]);

			String str = "abhiraj"; //null
			System.out.println(str.toUpperCase());
		} catch (ArithmeticException a) {
			System.out.println("Arithmetic Exception");
		} catch (ArrayIndexOutOfBoundsException o) {
			System.out.println("Array IndexOutOf BoundsException");
		} catch (Exception e) {
			System.out.println("Super Exception");
		}

	}

}
```

Output:

```
2
Array Exception
```

Example of Creating Our Own Exceptions:

```java
import java.util.Scanner;

public class ExceptionCreatingOurOwnExceptions {
	int balance;

	public void amountWithdraw(int a) {
		try {
			if (a > balance)
				throw new LessBalance(a, balance);// This is user defined

										// Exception LessBalance
			balance = balance - a;
			System.out.println("Amount Withdraw Your Balance is  " + balance);
		} catch (LessBalance e) {
			System.out.println("Enter Amount less Than " + balance);
		}
	}

	public static void main(String[] args) {

ExceptionCreatingOurOwnExceptions ob = new ExceptionCreatingOurOwnExceptions();

		Scanner c = new Scanner(System.in);
		System.out.println("Enter the Balance:");
		ob.balance = c.nextInt();

		System.out.println("Enter the Amount to WithDraw");
		int a = c.nextInt();

		ob.amountWithdraw(a);
	}
}
class LessBalance extends Exception {
	ExceptionCreatingOurOwnExceptions ob = new ExceptionCreatingOurOwnExceptions();

	public LessBalance() {

	}
	public LessBalance(int a, int b) {
		System.out.println("your balance is " + b + " and u want to withdraw "+ a);
	}
}
```

Output:

```
Enter the Balance:
1000
Enter the Amount to WithDraw
1500
your balance is 1000 and u want to withdraw 1500
Enter Amount less Than 1000
```