Advanced Programming with Java

Prativa Nyaupane

What we have learned..

- Setting up Java
- Setting up IDE
- Classes
- Objects
- Constructors
- Packages

Today’s Objective

- Variables and Data Types
- Conditional Statements
- Access Modifiers
- Exception Handling
- Java Collections

Variables

A variable is a container which holds the value, while the Java program is executed.

A variable is assigned with a data type.

A variable is the name of the memory location.

It is a combination of “vary + abled” - its value can be changed.

```java
int rollNumber1 = 1;
```

Types of variables

- **Local Variable**

  A variable declared inside the body of a method.

  Used only locally. Other methods do not have access to this variable.

- **Instance Variable**

  Declared inside the class but outside of body of a method.

  Its value is instance-specific and not shared among instances.

- **Static Variable**

  Declared as static, shared among all the instances of the class

  Memory allocation happens only once when the class is loaded in the memory

```java
public class Student {

    private static int passingScore = 60;//global variable

    int rollNumber; //instance(class) variable
    int score;//instance(class) variable


    public Student(int rollNumber, int score) {
        this.rollNumber = rollNumber;
        this.score = score;
    }


    public void checkIfStudentHasPassedOrNot() {
        boolean hasPassed = false;//local variable, can only be used by this method
        if (score >= passingScore) {
            hasPassed = true;

        } else {
            hasPassed = false;
        }

        System.out.println("Student passed "+hasPassed);

    }
```

Data Types:

Type of values that can be stored in variables. Broadly categorized into primitive(Intrinsic) and non-primitive(Derived)

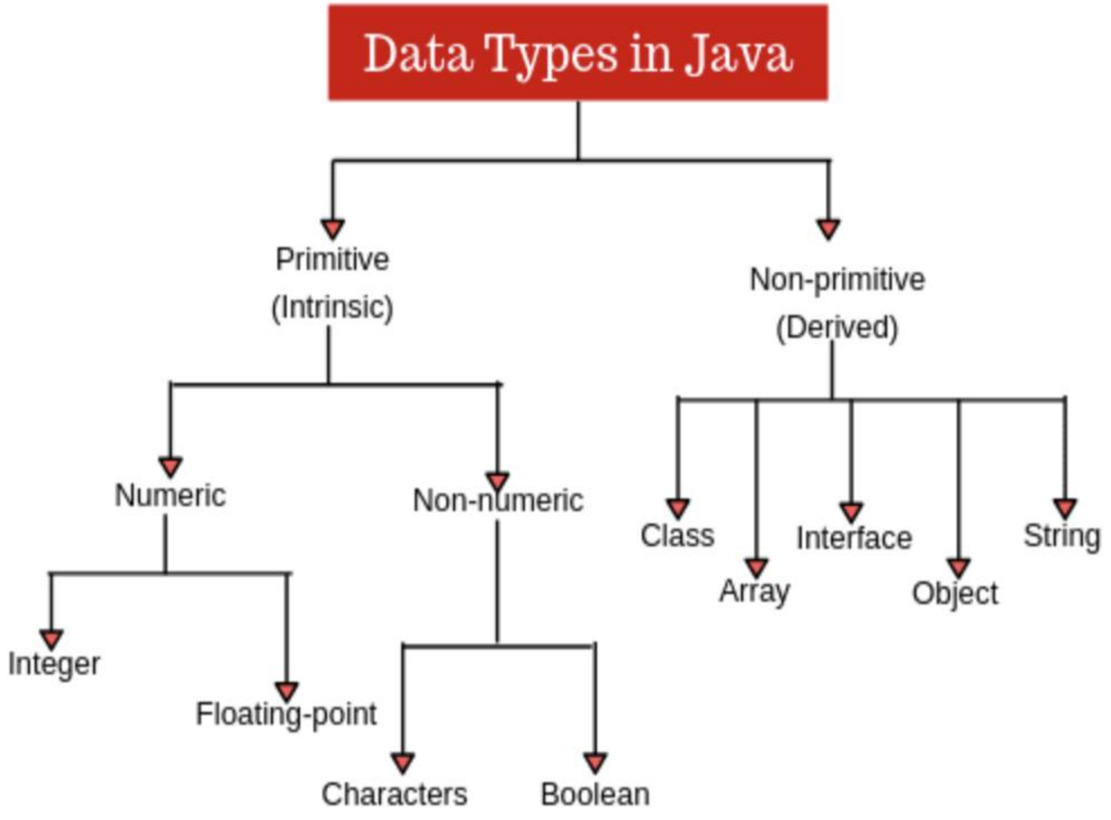


Fig: Classification of data types in java

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’,’\n’,’ β’	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Non-Primitive Data Types

Data types that do not hold a simple value directly but instead reference a memory location where the data is stored.

Created using classes, interfaces, arrays, and enums.

Classes	Interfaces	Arrays	Enums
Classes are user-defined data types that define the structure and behavior of objects.	Interfaces are reference types that define a set of methods that a class must implement.	Arrays are used to store multiple values of the same data type in a single variable.	Enums are a special type of class used to represent a fixed set of constants. Each constant value in an enum is an object of the enum type.
`class Person { String name; int age; }`	`interface Shape { void draw ); }`	`int[] numbers = {1, 2, 3, 4, 5};`	`enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY }`

Conditional Statements

Conditional Statements controls the flow of the execution of the program.

Java compiler executes the code from top to bottom.

The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of the Java code.

Types of Conditional Statements:

1. Decision Making statements
   - if statements
   - switch statement
2. Loop statements
   - for loop
   - for-each loop
3. Jump statements
   - break statement
   - continue statement

Decision Making Statements

```
public class Grade {

   public static int passMarks = 50;
   int marksObtained;

   public Grade(int marksObtained) {
       this.marksObtained = marksObtained;
       if (marksObtained >= passMarks) {
           handlePass(marksObtained);
       } else {
           handleFail();
       }
   }
public void handlePass(int marksObtained) {
   int grade = marksObtained / 10;
   switch (marksObtained) {
       case 10:
           System.out.println("Grade A");
```

```
break;
                  case 9:
                       System.out.println("Grade A");
                       break;
                  case 8:
                       System.out.println("Grade B");
                       break;
                  case 7:
                       System.out.println("Grade C");
                       break;
                  default:
                       System.out.println("Grade F");
              }
         }

         public void handleFail() {
              System.out.println("Student Failed");
         }}
```

Loop Statements

Find sum of numbers from 1 to 10 using for loop

```
public void findSumOfNumbers(){
    int sum = 0;
    for(int i = 1; i <=10; i++){
        sum = sum + i;
    }
}
```

Find sum of numbers from 1 to 10 using for-each loop

```
public void findSumOfNumbersUsingForEach() {
    int[] numbersToSum = {1, 2, 3, 4, 5, 6, 7,
8, 9, 10};
    int sum = 0;

    for(int number : numbersToSum){
        sum = sum + number;
    }


}
```

Access Modifiers

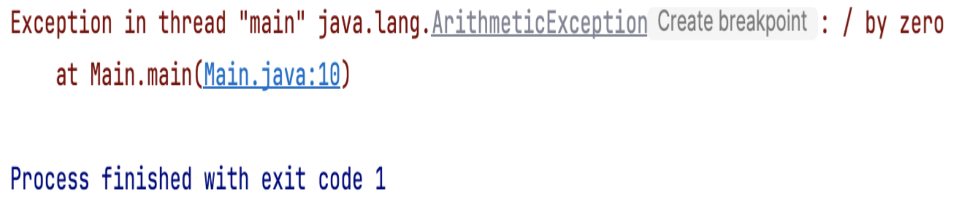
- Java provides access modifiers to control the visibility and accessibility of classes, variables, and methods.
- The access modifiers are "public", "protected", "private", and the default (no modifier).
- "Public" allows access from anywhere, "protected" allows access within the same package or subclasses, "private" restricts access to within the class itself, and the default restricts access to the same package.
- Access protection mechanisms ensure encapsulation, data hiding, and proper separation of concerns in an object-oriented system.

Exception Handling

- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.
- In Java, an exception is an event that disrupts the normal flow of the program.
- It is an object which is thrown at runtime.

```java
public class Main {
   public static void main(String[]
          args) {
       System.out.println(1 / 0);

   }}
```

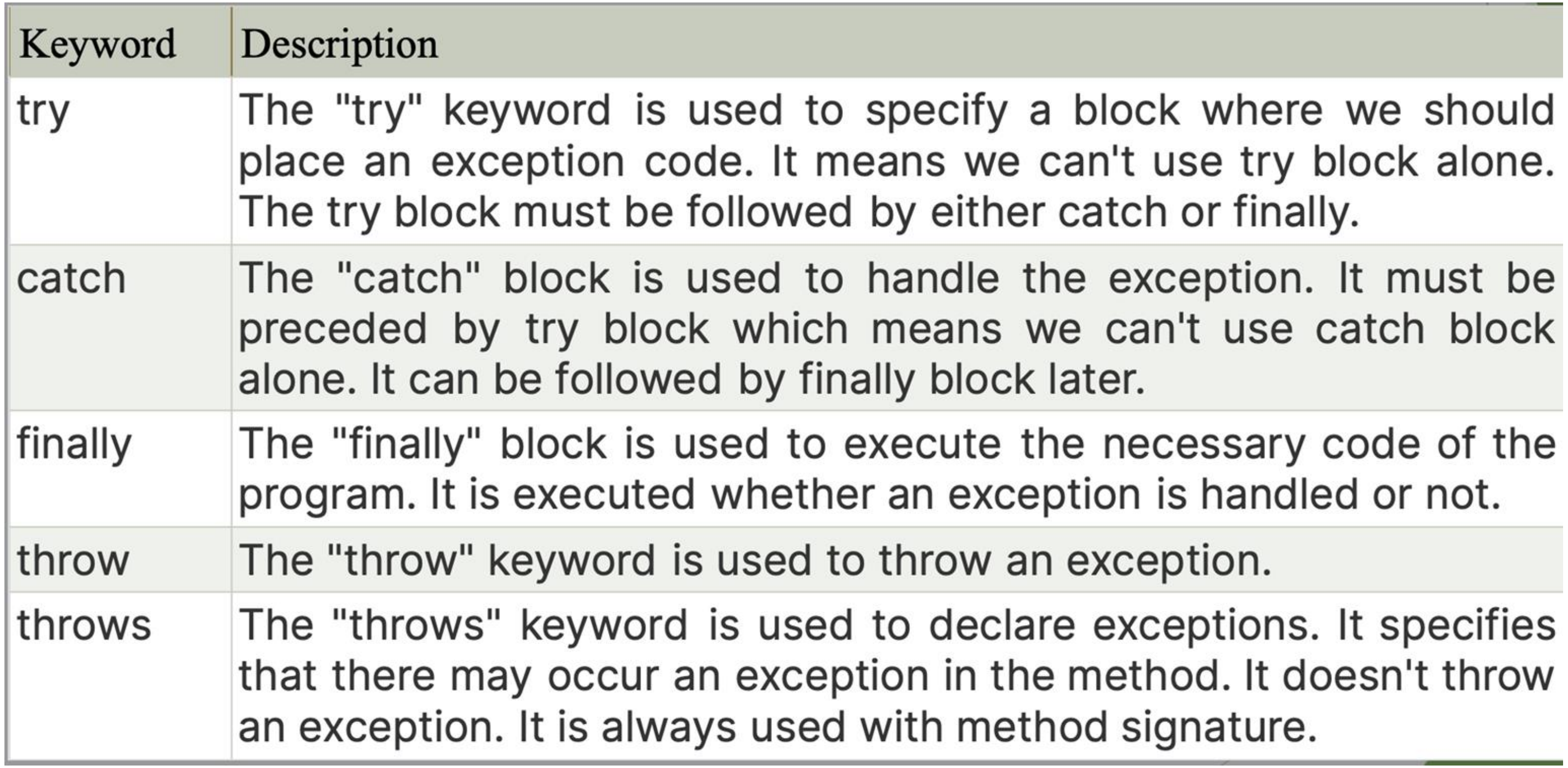


```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.main(Main.java:10)

Process finished with exit code 1
```

- The core advantage of exception handling is to maintain the normal flow of the application
- An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions

Exception Handling (contd...)



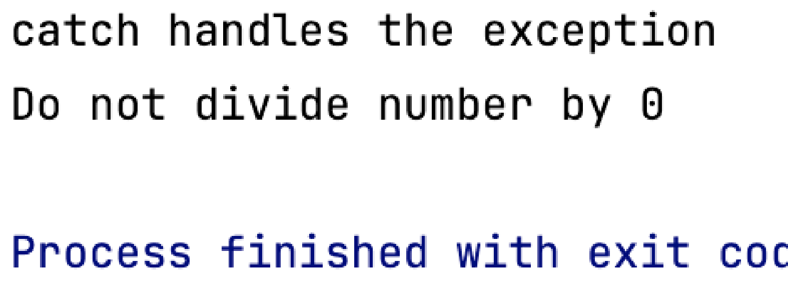
Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Exception Handling Contd..

```
public class Main {

   public static void main(String[] args) {
       try {
           System.out.println(1 / 0);
       }catch (ArithmeticException arithmeticException){
           System.out.println("catch handles the
exception");
       }finally {
           System.out.println("Do not divide
number by 0");
       }
   }

}
```



```
catch handles the exception
Do not divide number by 0

Process finished with exit coc
```

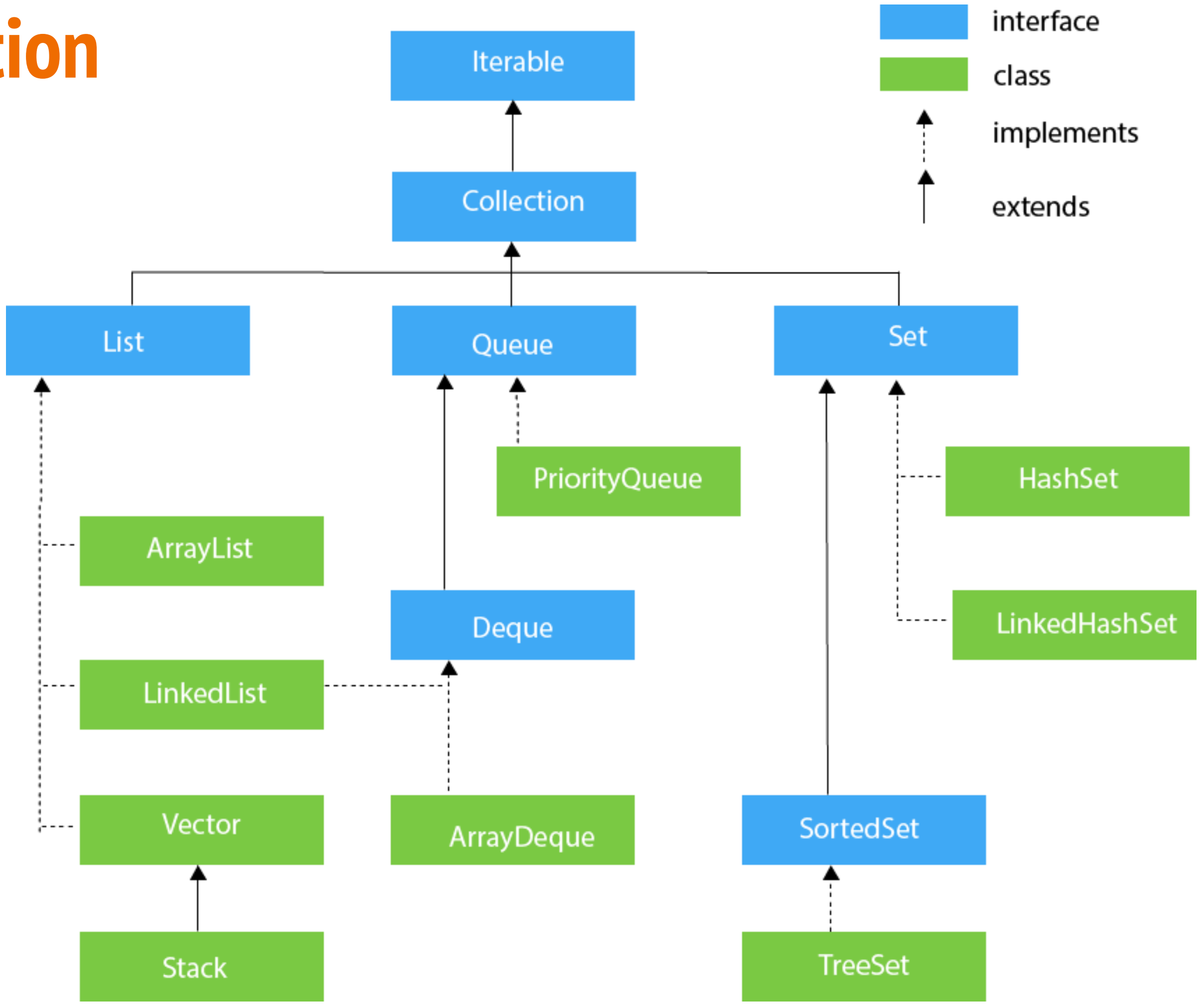
Collections

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

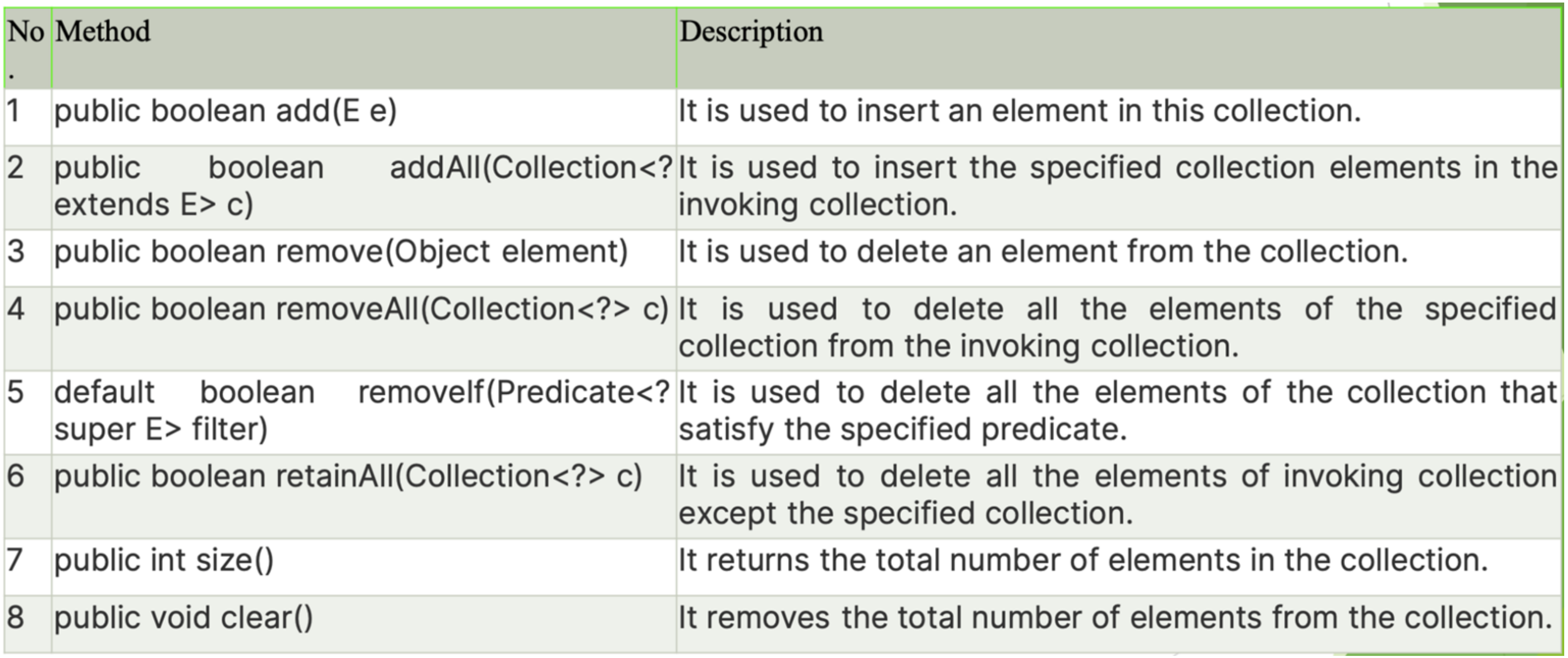
What is a framework in Java?

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

Hierarchy of Collection Framework



Methods of Collection Interface



No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.