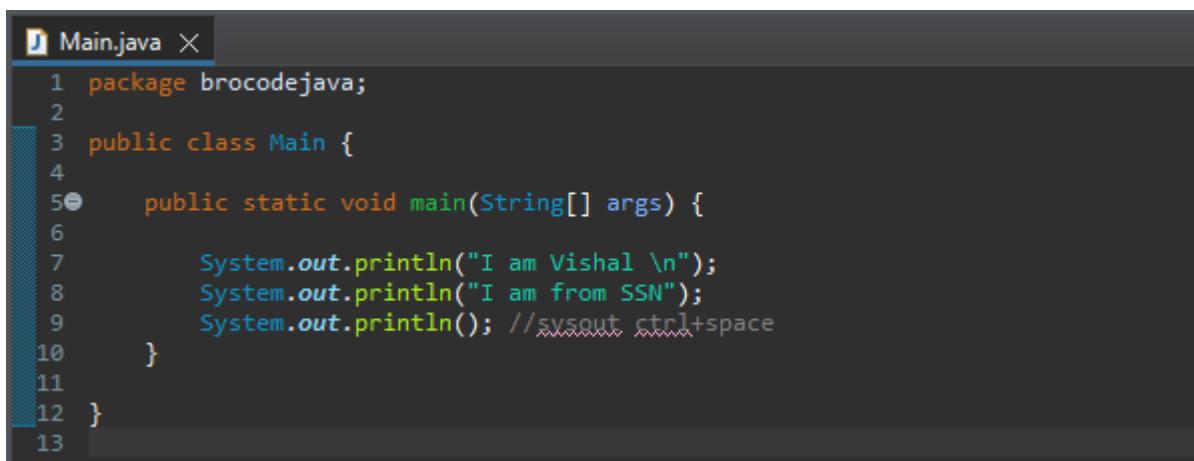


JAVA BROCODE

1.



```
Main.java
1 package brocodejava;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         System.out.println("I am Vishal \n");
8         System.out.println("I am from SSN");
9         System.out.println(); //sysout ctrl+space
10    }
11
12 }
13
```

2. Variables

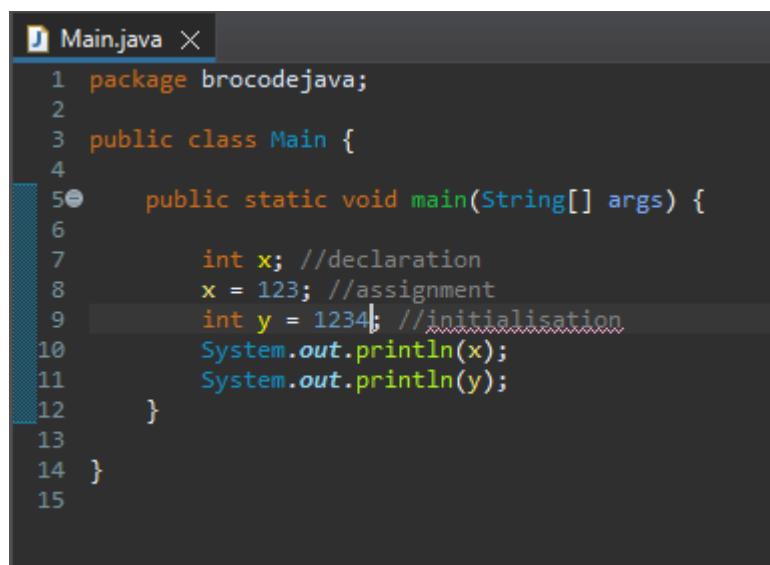
A placeholder for a value that behaves as the value it contains

<u>data type</u>	<u>size</u>	<u>primitive/ reference</u>	<u>value</u>
★ boolean	1 bit	primitive	true or false
byte	1 byte	primitive	-128 to 127
short	2 bytes	primitive	-32,768 to 32,767
★ int	4 bytes	primitive	-2 billion to 2 billion
long	8 bytes	primitive	-9 quintillion to 9 quintillion
float	4 bytes	primitive	fractional number up to 6-7 digits ex. 3.141592f
★ double	8 bytes	primitive	fractional number up to 15 digits ex. 3.141592653589793
★ char	2 bytes	primitive	single character/letter/ASCII value ex. 'f'
★ String	varies	reference	a sequence of characters ex. "Hello world!"

primitive vs reference

- 8 types (boolean, byte, etc.)
- stores data
- can only hold 1 value
- less memory
- fast
- unlimited (user defined)
- stores an address
- could hold more than 1 value
- more memory
- slower

variable → declaration + assignment or initialisation



The screenshot shows a Java code editor window titled "Main.java". The code is as follows:

```
1 package brocodejava;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         int x; //declaration
8         x = 123; //assignment
9         int y = 1234; //initialisation
10        System.out.println(x);
11        System.out.println(y);
12    }
13
14 }
```

Main.java

```
1 package brocodejava;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         long x = 1234567890L;
8         int y = 123;
9         float z = 3.14f;
10        boolean f = false;
11
12        char symbol = '@';
13        String name = "Vishal is atomic";
14
15        System.out.println("The number is "+x);
16        System.out.println("The number is "+y);
17        System.out.println("The number is "+z);
18        System.out.println(f);
19        System.out.println(symbol);
20        System.out.println("I know that "+name);
21    }
22
23 }
24
```

3. Swap two variables

*Three.java

```
1 package brocodejava;
2
3 public class Three {
4
5     public static void main(String[] args) {
6         String x = "water";
7         String y = "Acid";
8         String temp = null;
9
10        temp = x;
11        x=y;
12        y=temp;
13
14        System.out.println("x: "+x);
15        System.out.println("y: "+y);
16    }
17
18 }
19
```

4. User Input

```
Four.java ×
1
2 package brocodejava;
3 import java.util.Scanner;
4 public class Four {
5
6     public static void main(String[] args) {
7
8         Scanner scanner = new Scanner(System.in);
9         System.out.println("What is your name? ");
10
11        String name = scanner.nextLine();
12        System.out.println("How old are you? ");
13        int age = scanner.nextInt();
14        scanner.nextLine(); //to clear the contents in the scanner
15        System.out.println("What is your favourite food? ");
16        String food = scanner.nextLine();
17
18        System.out.println("Hello " + name);
19        System.out.println("You are " + age + " years old");
20        System.out.println("You like " + food);
21        scanner.close();
22    }
23
24
25 }
26 |
```

```
<terminated> Four [Java Application]
What is your name?
Vishal
How old are you?
21
What is your favourite food?
lassi
Hello Vishal
You are 21 years old
You like lassi
```

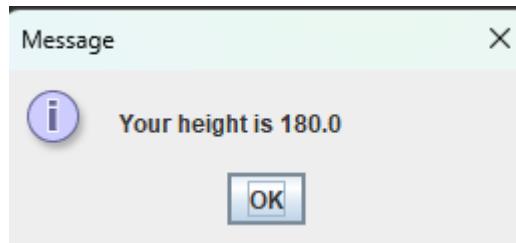
5. Expressions

```
Five.java X
1 package brocodejava;
2
3 public class Five {
4
5     public static void main(String[] args) {
6         // expression = operands & operators
7         // operands = values, variables, numbers, quantity
8         // operators = +-*%
9
10    //     int num = 10;
11    //     num = num/3;
12
13    double num = 10;
14    num = (double) num/3;
15
16    System.out.println(num);
17
18 }
19
20
21 }
22
```

6. GUI intro

```
Six.java X
1 package brocodejava;
2 import javax.swing.JOptionPane;
3 public class Six {
4
5     public static void main(String[] args) {
6         String name = JOptionPane.showInputDialog("Enter your name ");
7
8         JOptionPane.showMessageDialog(null, "Hello "+name);
9
10        int age = Integer.parseInt(JOptionPane.showInputDialog("Enter your age: "));
11        JOptionPane.showMessageDialog(null, "Your age is "+age);
12
13        double height = Double.parseDouble(JOptionPane.showInputDialog("Enter your height: "));
14        JOptionPane.showMessageDialog(null, "Your height is "+height);
15
16    }
17
18 }
```





7. Math Class

```
1 package brocodejava;
2 import java.util.Scanner;
3 public class Seven {
4
5     public static void main(String[] args) {
6
7         double x = 3.14;
8         double y = -10;
9         double z = Math.max(x,y);
10        double z = Math.min(x,y);
11        double z = Math.abs(y);
12        double z = Math.sqrt(x);
13        double z = Math.round(x);
14        double z = Math.ceil(x);
15        double z = Math.floor(x);
16        System.out.println(z);
17
18        double x;
19        double y;
20        double z;
21
22        Scanner scanner = new Scanner(System.in);
23
24        System.out.println("Enter side x: ");
25        x = scanner.nextDouble();
26        System.out.println("Enter side y: ");
27        y = scanner.nextDouble();
28
29        z = Math.sqrt((x*x)+(y*y));
30        System.out.println("Hypotenuse: "+z);
31
32    }
33
34 }
35 }
```

A screenshot of a Java code editor showing a file named "Seven.java". The code uses the Math class to calculate the hypotenuse of a right-angled triangle given two sides. It includes imports for the Scanner class, defines a Seven class with a main method, and uses various Math methods like max, min, abs, sqrt, round, ceil, and floor.

8. Random Numbers

```
Eight.java X
1 package brocodejava;
2 import java.util.Random;
3 public class Eight {
4
5     public static void main(String[] args) {
6
7         Random random = new Random();
8
9 //         int x = random.nextInt(); (Result will be in the range -2billion to positive 2billion)
10 //        int x = random.nextInt(6)+1;
11
12 //        double y = random.nextDouble(); (Random value between 0 and 1)
13 //        boolean z = random.nextBoolean();
14
15         System.out.println(z);
16     }
17
18 }
19
```

9. if statements

```
Nine.java X
1 package brocodejava;
2
3 public class Nine {
4
5     public static void main(String[] args) {
6
7         int age = 80;
8
9         if (age>=75) {
10             System.out.println("Ok Boomer");
11         }
12         else if(age>=18) {
13             System.out.println("You are an adult");
14         }
15         else {
16             System.out.println("You are not an adult");
17         }
18     }
19
20 }
21
22
```

10. switches

A screenshot of a Java code editor showing a file named "Ten.java". The code contains a main method that reads a day from the user and prints it out. A switch statement is used to handle each day of the week. The code is as follows:

```
1 package brocodejava;
2 import java.util.Scanner;
3 public class Ten {
4
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7         System.out.println("Enter the Day");
8         String day = scanner.nextLine();
9
10        switch(day) {
11            case "Sunday": System.out.println("It is Sunday");
12            break;
13            case "Monday": System.out.println("It is Monday");
14            break;
15            case "Tuesday": System.out.println("It is Tuesday");
16            break;
17            case "Wednesday": System.out.println("It is Wednesday");
18            break;
19            case "Thursday": System.out.println("It is Thursday");
20            break;
21            case "Friday": System.out.println("It is Friday");
22            break;
23            case "Saturday": System.out.println("It is Saturday");
24            break;
25            default: System.out.println("Not a day");
26        }
27    }
28}
29
30}
```

11. Logical Operators

```
Eleven.java ×
1 package brocodejava;
2
3 public class Eleven {
4
5     public static void main(String[] args) {
6         //      && = AND : Both conditions must be true
7         //      || = OR : either condition must be true
8         //      ! = NOT : reverses boolean value of a condition
9
10        int temp = 25;
11
12        if (temp>30) {
13            System.out.println("It is hot outside");
14        }
15        else if(temp>=20 && temp<=30) {
16            System.out.println("It is warm outside");
17        }
18        else {
19            System.out.println("It is cold outside");
20        }
21    }
22}
23
24}
```

```
12
13     Scanner scanner = new Scanner(System.in);
14
15     System.out.println("You are playing a game! Press q or Q to quit");
16     String response = scanner.next();
17
18     if(!response.equals("q") && !response.equals("Q")) {
19         System.out.println("You are still playing the game *pew pew*");
20     }
21     else {
22         System.out.println("You quit the game");
23     }
24
25 }
```

Problems Javadoc Declaration Console Coverage Call Hierarchy

<terminated> Main (6) [Java Application] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (Oct 14, 2020, 4:26:15 PM)

```
You are playing a game! Press q or Q to quit
Q
You quit the game
```

12. while loop

```
1 package brocodejava;
2 import java.util.Scanner;
3 public class Twelve {
4
5     public static void main(String[] args) {
6         // as long as the condition is true
7         Scanner scanner = new Scanner(System.in);
8         String name = "";
9
10        while(name.isBlank()) {
11            System.out.println("Enter your name: ");
12            name = scanner.nextLine();
13        }
14
15        do {
16            System.out.println("Enter your name:");
17            name = scanner.nextLine();
18        } while (name.isBlank());
19
20
21        System.out.println("Hello "+name);
22
23    }
24
25 }
26
```

13. For Loop

```
1 package brocodejava;
2 import java.util.Scanner;
3 public class Thirteen {
4
5     public static void main(String[] args) {
6
7         for (int i = 0; i<=10;i++) {
8             System.out.println(i);
9         }
10
11     }
12
13 }
14
```

14. Nested Loops

```
Fourteen.java ×
1 package brocodejava;
2 import java.util.Scanner;
3 public class Fourteen {
4
5     public static void main(String[] args) {
6
7         Scanner scanner = new Scanner(System.in);
8         int rows;
9         int columns;
10        String symbol = "";
11
12        System.out.println("Enter number of rows");
13        rows = scanner.nextInt();
14
15        System.out.println("Enter number of cols");
16        columns = scanner.nextInt();
17
18        System.out.println("Enter the symbol");
19        symbol = scanner.next();
20
21        for (int i=1;i<=rows;i++) {
22            System.out.println();
23            for (int j=1;j<=columns;j++) {
24                System.out.print(symbol);
25            }
26        }
27
28    }
29
30 }
31
```

15. Arrays

The screenshot shows a Java development environment with two windows. The top window is titled 'Fifteen.java' and contains the following Java code:

```
1 package brocodejava;
2
3 public class Fifteen {
4
5     public static void main(String[] args) {
6
7         String[] cars = {"Camaro", "Tesla", "Santro"};
8         int[] numbers = {1,2,3,4};
9         cars[0] = "Mustang";
10        System.out.println(cars[0]);
11        System.out.println(numbers[3]);
12
13        String[] arr = new String[3];
14
15        arr[0] = "Ertiga";
16        arr[1] = "Luxus";
17        arr[2] = "Carens";
18
19        System.out.println(arr[0]);
20
21        for (int i=0;i<arr.length;i++) {
22            System.out.println(arr[i]);
23        }
24
25    }
26
27
28 }
29
```

The bottom window is titled 'Console' and displays the output of the program:

```
<terminated> Fifteen [Java Application] C:\Program Files\Java\jdk-22\bin\javaw
Mustang
4
Ertiga
Ertiga
Luxus
Carens
```

16. 2D Arrays

The screenshot shows a Java application running in an IDE. The code editor window is titled "Sixteen.java" and contains the following Java code:

```
1 package brocodejava;
2
3 public class Sixteen {
4
5     public static void main(String[] args) {
6
7         // 2D arrays = array of arrays
8
9         String[][] cars = new String[3][3];
10
11        cars[0][0] = "Camaro";
12        cars[0][1] = "Corvette";
13        cars[0][2] = "Ertiga";
14        cars[1][0] = "Alto";
15        cars[1][1] = "Ritz";
16        cars[1][2] = "Nano";
17        cars[2][0] = "Carnival";
18        cars[2][1] = "Carens";
19        cars[2][2] = "Mustang";
20
21        for (int i=0; i<cars.length;i++) {
22            System.out.println();
23            for (int j=0;j<cars[i].length;j++) {
24                System.out.print(cars[i][j]+ " ");
25            }
26
27        String[][] cars1 = {{"Camaro","Corvette","Ertiga"},}
28                    {"Alto","Ritz","Nano"},}
29                    {"Carnival","Carens","Mustang"}};
30
31        System.out.println();
32        for (int i1=0; i1<cars1.length;i1++) {
33            System.out.println();
34            for (int j=0;j<cars1[i1].length;j++) {
35                System.out.print(cars1[i1][j]+ " ");
36            }
37        }
38
39    }
40
41 }
42 }
```

The terminal window below the code editor shows the output of the program:

```
<terminated> Sixteen [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (19 Au
Alto Ritz Nano
Carnival Carens Mustang
Alto Ritz Nano

Camaro Corvette Ertiga
Alto Ritz Nano
Carnival Carens Mustang
Carnival Carens Mustang

Camaro Corvette Ertiga
Alto Ritz Nano
Carnival Carens Mustang
```

17. String Methods

```
1 *Seventeen.java ×
2
3 public class Seventeen {
4
5     public static void main(String[] args) {
6         // String = a reference data type that can store one or more characters
7         //           reference data types have access to useful methods
8         String name = "Vishal";
9         boolean result = name.equals("Vishal");
10        boolean result1 = name.equals("vishal");
11        boolean result2 = name.equalsIgnoreCase("vishal");
12
13        if (name.equals(" Vishal ")) {
14            System.out.println("True Vishal Real");
15        }
16        System.out.println(result);
17        System.out.println(result1);
18        System.out.println(result2);
19        System.out.println();
20
21        int result3 = name.length();
22        System.out.println(result3);
23        System.out.println();
24
25        char result4 = name.charAt(3);
26        System.out.println(result4);
27        System.out.println();
28
29        int result5 = name.indexOf('a');
30        System.out.println(result5);
31        System.out.println();
32
33        int result6 = name.indexOf('z');
34        System.out.println(result6);
35        System.out.println();
36
37        boolean result7 = name.isEmpty();
38        System.out.println(result7);
39        System.out.println();
40
41        String result8 = name.toUpperCase();
42        System.out.println(result8);
43
44        String result9 = name.toLowerCase();
45        System.out.println(result9);
46        System.out.println();
47
48        String result10 = name.trim();
49        System.out.println(result10);
50        System.out.println();
51
52        String result11 = name.replace("Vi","Rizz");
53        System.out.println(result11);
54
```

```
<terminated>
true
false
true

6

h

4

-1

false

VISHAL
vishal

Vishal

Rizzshal
```

18. Wrapper Classes

```
// wrapper class = provides a way to use primitive data types as reference data types
//                         reference data types contain useful methods
//                         can be used with collections (ex.ArrayList)

//primitive      //wrapper
//-----      -----
// boolean      Boolean
// char        Character
// int          Integer
// double       Double

// autoboxing = the automatic conversion that the Java compiler makes between the primitive t
// unboxing = the reverse of autoboxing. Automatic conversion of wrapper class to primitive
```

```
1 package brocodejava;
2
3 public class Eighteen {
4
5     public static void main(String[] args) {
6         // wrapper class = provides a way to use primitive data types as reference data types
7         // reference data types contain useful methods
8         // can be used with collections
9
10        // autoboxing = the automatic conversion that the java compiler makes between the primitive types
11        // and their corresponding object wrapper classes
12        // unboxing = the reverse of autoboxing. Automatic conversion of wrapper class to primitive type
13
14        //autoboxing
15        Boolean a = true;
16        Character b = '@';
17        Integer c = 1234;
18        Double d = 3.14;
19        String e = "Vishal";
20
21        //unboxing
22        if (a == true) {
23            System.out.println("This is true");
24        }
25
26    }
27
28 }
29
```

19. ArrayList

```
1 package brocodejava;
2 import java.util.ArrayList;
3 public class Nineteen {
4
5     public static void main(String[] args) {
6         // ArrayList = resizable array
7         // Elements can be added and removed after compilation phase
8         // store reference data types
9         // To declare integer array, use wrapper class instead of ArrayList<int>. must be ArrayList<Integer>
10        ArrayList<String> food = new ArrayList<String>();
11
12        food.add("pizza");
13        food.add("burger");
14        food.add("lassi");
15
16        food.set(0, "Sushi");
17        food.remove(2);
18        food.clear();
19        for(int i=0;i<food.size();i++) {
20            System.out.println(food.get(i));
21        }
22
23
24
25
26    }
27
28 }
29
```

20. 2D ArrayList

The screenshot shows a Java development environment with two tabs open: "Twenty.java" and "Console".

Twenty.java:

```
1 package brocodejava;
2 import java.util.*;
3 public class Twenty {
4
5     public static void main(String[] args) {
6
7         ArrayList<ArrayList<String>> groceryList = new ArrayList();
8
9
10        ArrayList<String> bakeryList = new ArrayList();
11        bakeryList.add("Pasta");
12        bakeryList.add("Garlic Bread");
13        bakeryList.add("Donuts");
14
15        System.out.println(bakeryList);
16        System.out.println(bakeryList.get(0));
17
18        ArrayList<String> produceList = new ArrayList();
19        produceList.add("Tomatoes");
20        produceList.add("zucchini");
21        produceList.add("peppers");
22
23        System.out.println(produceList);
24        System.out.println(produceList.get(0));
25
26        groceryList.add(bakeryList);
27        groceryList.add(produceList);
28
29        System.out.println(groceryList);
30        System.out.println(groceryList.get(0));
31        System.out.println(groceryList.get(0).get(0));
32
33    }
34
35
36 }
37
```

Console:

```
<terminated> Twenty [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (19 Aug 2024, 11:26:21)
[Pasta, Garlic Bread, Donuts]
Pasta
[Tomatoes, zucchini, peppers]
Tomatoes
[[Pasta, Garlic Bread, Donuts], [Tomatoes, zucchini, peppers]]
[Pasta, Garlic Bread, Donuts]
Pasta
```

21. for-each loop

```
TwentyOne.java ×
1 package brocodejava;
2
3 import java.util.ArrayList;
4
5 public class TwentyOne {
6
7     public static void main(String[] args) {
8
9         //String[] animals = {"cat","dog","rat","bird"};
10        ArrayList<String> animals = new ArrayList<String>();
11        animals.add("dog");
12        animals.add("cat");
13        animals.add("rat");
14        animals.add("bird");
15        for(String i : animals) {
16            System.out.println(i);
17        }
18    }
19}
20
21}
22
```

22. Methods

```
TwentyTwo.java ×
1 package brocodejava;
2
3 public class TwentyTwo {
4
5     public static void main(String[] args) {
6         // method = block of code which is executed whenever it is called upon
7         String name = "Vishal";
8         int age = 21;
9
10        hello(name,age);
11
12    }
13
14
15    static void hello(String name, int age) {
16        System.out.println("Hello "+name);
17        System.out.println("Your are "+age);
18    }
19}
20
21}
22
```

```
TwentyTwoagain.java ×
1 package brocodejava;
2
3 public class TwentyTwoagain {
4
5     public static void main(String[] args) {
6
7         int x = 3;
8         int y = 4;
9
10        int z = add(x,y);
11        System.out.println(z);
12    }
13
14
15    static int add(int x, int y) {
16        int z = x+y;
17        return z;
18    }
19 }
20
```

23. Overloaded Methods

```

1 *TwentyThree.java ×
2
3 public class TwentyThree {
4
5     public static void main(String[] args) {
6         // overload methods = methods that share the same name but have different parameters
7         //                                     method name + parameters = method signature
8         //                                     parameters = number of parameters, data type and the order of parameters can be changed
9         //                                     and overload methods can be created with same name but different parameters
10
11     int x = add(1,2);
12     System.out.println(x);
13
14     int y = add(1,2,3);
15     System.out.println(y);
16
17     int z = add(1,2,3,4);
18     System.out.println(z);
19
20     double x1 = add(1.1,2.2);
21     System.out.println(x1);
22
23     double y1 = add(1.1,2.2,3.3);
24     System.out.println(y1);
25
26     double z1 = add(1.1,2.2,3.3,4.4);
27     System.out.println(z1);
28 }
29
30     static int add(int a,int b) {
31         System.out.println("This is overload method #1");
32         return a+b;
33 }
34
35     static int add(int a,int b,int c) {
36         System.out.println("This is overload method #2");
37         return a+b+c;
38 }
39
40     static int add(int a,int b,int c, int d) {
41         System.out.println("This is overload method #3");
42         return a+b+c+d;
43 }
44
45     static double add(double a,double b) {
46         System.out.println("This is overload method #4");
47         return a+b;
48 }
49
50     static double add(double a,double b,double c) {
51         System.out.println("This is overload method #5");
52         return a+b+c;
53 }

```

```

<terminated> TwentyThree [Java App]
This is overload method #1
3
This is overload method #2
6
This is overload method #3
10
This is overload method #4
3.3000000000000003
This is overload method #5
6.6
This is overload method #6
11.0

```

24. printf

```
TwentyFour.java ×
3  public class TwentyFour {
4
5    public static void main(String[] args) {
6        // printf() = optional method to control,format and display test to the console window
7        //           two arguments = format string + (object/variable/value)
8        //           % [flags] [precision] [width] [conversion-character]
9
10       System.out.printf("%s is a format string %d ","vishal",123);
11       boolean myBoolean = true;
12       char myChar = '@';
13       String myString = "Vishal";
14       int myInt = 50;
15       double myDouble = 34589.145246;
16
17       // [conversion-character]
18       System.out.printf("%b",myBoolean);
19       System.out.println();
20       System.out.printf("%c",myChar);
21       System.out.println();
22       System.out.printf("%s",myString);
23       System.out.println();
24       System.out.printf("%d",myInt);
25       System.out.println();
26       System.out.printf("%f",myDouble);
27       System.out.println();
28       // [width]
29       // minimum number of characters to be written as output
30       System.out.printf("Hello %10s", myString);
31       System.out.println();
32       // [precision]
33       // sets number of digits of precision when outputting floating-point values
34
35       System.out.printf("You have %.2f money left",myDouble);
36       System.out.println();
37       System.out.printf("You have %f money left",myDouble);
38       System.out.println();
39
40       // [flags]
41       // adds an effect to output based on the flag added to format specifier
42       // - : left-justify
43       // + : output a plus (+) or minus(-) sign for a numeric value
44       // 0 : numeric values are zero-padded
45       // , : comma grouping separator if numbers>1000
46       System.out.println();
47       System.out.printf("You have %-10f money left",myDouble);
48       System.out.println();
49       System.out.printf("You have %+f money left",myDouble);
50       System.out.println();
51       System.out.printf("You have %020f money left",myDouble);
52       System.out.println();
53       System.out.printf("You have %,f money left",myDouble);
54
55   }
```

```
vishal is a format string 123 true
@
Vishal
50
34589.145246
Hello      Vishal
You have 34589.15 money left
You have 34589.145246 money left

You have 34589.145246 money left
You have +34589.145246 money left
You have 0000000034589.145246 money left
You have 34,589.145246 money left
```

25. final keyword

The screenshot shows a Java code editor and a terminal window. The code editor has a dark theme and displays the following Java code:

```
1 package brocodejava;
2
3 public class TwentyFive {
4
5     public static void main(String[] args) {
6         // anything that has been declared final cannot be updated or changed later
7         final double PI = 3.14159;
8         PI = 4;
9
10        System.out.println(PI);
11    }
12
13 }
14
15 }
```

The line `PI = 4;` is highlighted in red, indicating a syntax error. The terminal window below shows the compilation output:

```
<terminated> TwentyFive [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (20 Aug 2024, 8:13:59 pm – 8:13:59 pm) [pid: 15848]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The final local variable PI cannot be assigned. It must be blank and not using a compound assignment
at brocodejava.TwentyFive.main(TwentyFive.java:8)
```

26. Object Oriented Programming (OOP)

```
TwentySix.java X Car.java
1 package brocodejava;
2
3 public class TwentySix {
4
5     public static void main(String[] args) {
6
7         // object = an instance of a class that may contain attributes and methods
8         // example: phone, desk, computer, coffee cup
9
10        Car myCar1 = new Car();
11        Car myCar2 = new Car();
12
13        System.out.println(myCar1.model);
14        System.out.println(myCar1.make);
15        System.out.println();
16        System.out.println(myCar2.model);
17        System.out.println(myCar2.make);
18
19        myCar1.drive();
20        myCar1.brake();
21
22    }
23
24 }
25
26
27
```

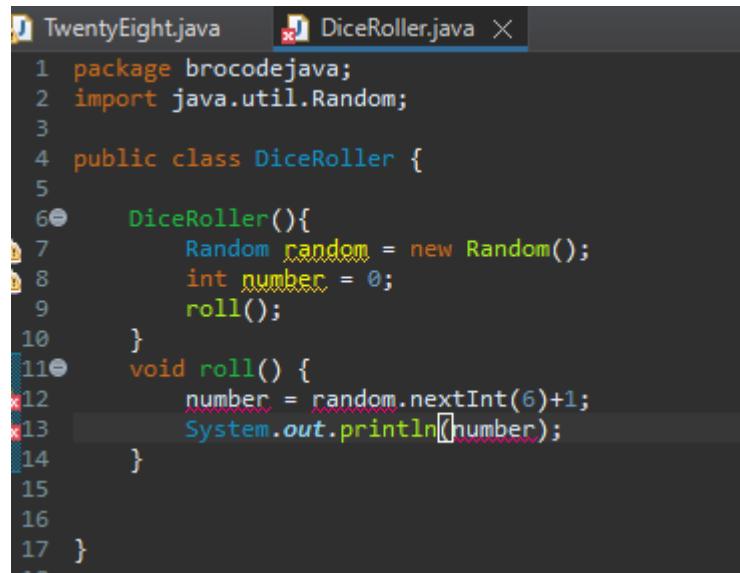
```
Car.java X
1 package brocodejava;
2
3 public class Car {
4
5     String make = "Suzuki";
6     String model = "Ertiga";
7     int year = 2020;
8     String colour = "Blue";
9     double price = 500000.00;
10
11
12     void drive() {
13         System.out.println("You are driving the car");
14     }
15
16     void brake() {
17         System.out.println("You have pressed the brake!");
18     }
19
20 }
21
```

27. Constructors

```
TwentySeven.java X
1 package brocodejava;
2
3 public class TwentySeven {
4
5     public static void main(String[] args) {
6         // constructors = a special method that is called when the object is instantiated (created)
7
8         Human human1 = new Human("Vishal",21,67.4);
9         Human human2 = new Human("Raju",21,47.4);
10
11         System.out.println(human1.name);
12         System.out.println(human2.name);
13
14         human2.eat();
15         System.out.println();
16         human1.drink();
17     }
18 }
19 class Human{
20     String name;
21     int age;
22     double weight;
23     //constructors
24     Human(String name, int age,double weight){
25         this.name = name;
26         this.age = age;
27         this.weight = weight;
28     }
29
30     void eat() {
31         System.out.printf("%s is eating",this.name);
32     }
33     void drink() {
34         System.out.println(this.name+" is drinking");
35     }
36 }
```

```
Console X
<terminated> TwentySeven [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (20 Aug 2024, 10:32:07 pm – 10:32:07 pm) [pid: 944]
Vishal
Raju
Raju is eating
Vishal is drinking
```

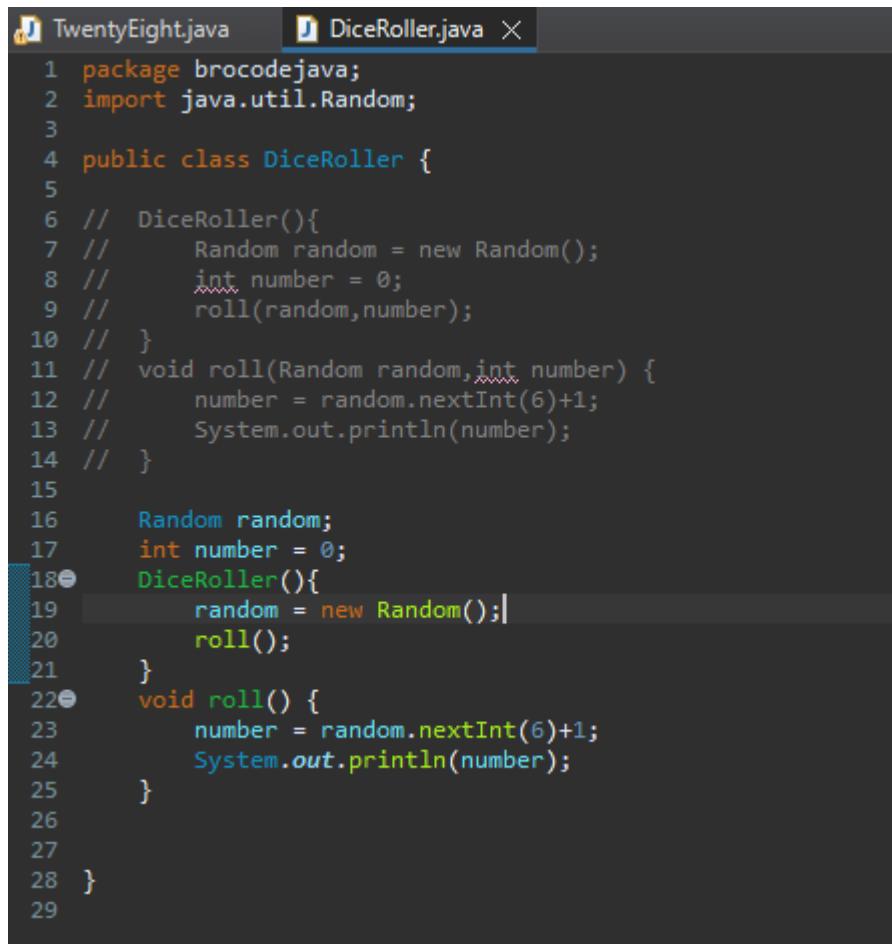
28. Variable Scope



```
TwentyEight.java DiceRoller.java X
1 package brocodejava;
2 import java.util.Random;
3
4 public class DiceRoller {
5
6     DiceRoller(){
7         Random random = new Random();
8         int number = 0;
9         roll();
10    }
11    void roll() {
12        number = random.nextInt(6)+1;
13        System.out.println(number);
14    }
15
16
17 }
```

Random and number only visible to the constructor method

to resolve, either pass them as instance through arguments or declare them global



```
TwentyEight.java DiceRoller.java X
1 package brocodejava;
2 import java.util.Random;
3
4 public class DiceRoller {
5
6     // DiceRoller(){
7     //     Random random = new Random();
8     //     int number = 0;
9     //     roll(random,number);
10    //}
11    // void roll(Random random,int number) {
12    //     number = random.nextInt(6)+1;
13    //     System.out.println(number);
14    //}
15
16    Random random;
17    int number = 0;
18    DiceRoller(){
19        random = new Random();
20        roll();
21    }
22    void roll() {
23        number = random.nextInt(6)+1;
24        System.out.println(number);
25    }
26
27
28 }
29
```

The screenshot shows a Java code editor with two files open: `TwentyEight.java` and `DiceRoller.java`. The `TwentyEight.java` file contains the following code:

```
1 package brocodejava;
2
3 public class TwentyEight {
4
5     public static void main(String[] args) {
6         // local = declared inside a method
7         //           visible only to that method
8
9         // global = declared outside a method, but within a class
10        //           visible to all parts of a class
11
12        DiceRoller diceRoller = new DiceRoller();
13        |
14
15    }
16
17 }
```

29. Overloaded Constructors

```
TwentyNine.java ×
1 package brocodejava;
2
3 public class TwentyNine {
4
5     public static void main(String[] args) {
6         // overloaded constructors = multiple constructors within a class with the same name,
7         //                                but have different parameters
8         //                                name + parameters = signature
9
10        //    Pizza pizza = new Pizza("thicc crust", "tomato", "cheddar", "olives");
11        Pizza pizza = new Pizza("thicc crust", "tomato", "cheddar");
12        System.out.println("Here are the ingredients of your pizza");
13        System.out.println(pizza.bread);
14        System.out.println(pizza.sauce);
15        System.out.println(pizza.cheese);
16        //    System.out.println(pizza.topping);
17
18    }
19
20}
21 class Pizza{
22    String bread;
23    String sauce;
24    String cheese;
25    String topping;
26
27    Pizza(String bread, String sauce, String cheese){
28
29        this.bread = bread;
30        this.sauce = sauce;
31        this.cheese = cheese;
32
33    }
34
35    Pizza(String bread, String sauce, String cheese, String topping){
36
37        this.bread = bread;
38        this.sauce = sauce;
39        this.cheese = cheese;
40        this.topping = topping;
41
42    }
43}
```

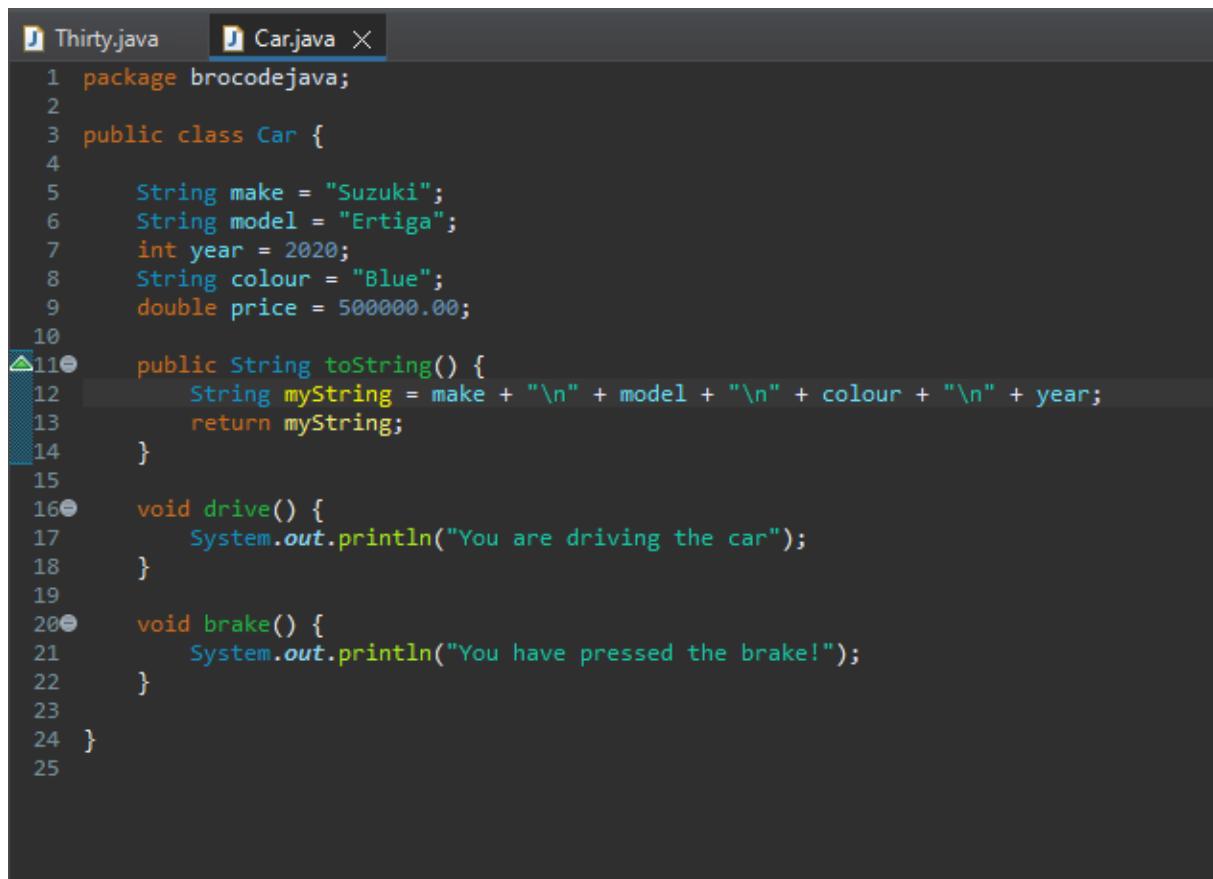


```
Console ×
<terminated> TwentyNine [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (20 Aug 2024, 11:04:29 pm - 11:04:29 pm) [pid: 23808]
Here are the ingredients of your pizza
thicc crust
tomato
cheddar
```

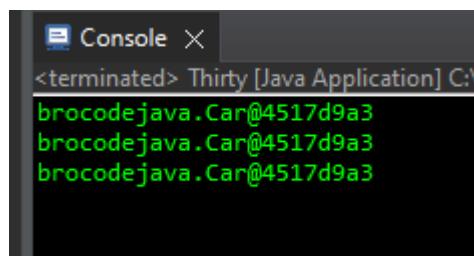
30. `toString` method

```
Thirty.java X Car.java
1 package brocodejava;
2
3 public class Thirty {
4
5     public static void main(String[] args) {
6         // toString() = special method that all objects inherit,
7         //             that returns a string that "textually represents" an object.
8         //             can be used both implicitly and explicitly
9
10        Car car = new Car();
11
12        System.out.println(car.make);
13        System.out.println(car.model);
14        System.out.println(car.colour);
15        System.out.println(car.year);
16        System.out.println(car); //displays the address of the object, implicitly calls toString() method
17        //the above line is equivalent to
18        System.out.println(car.toString());
19
20        //however after method overriding, i.e., declaring a method toString in the Car class, the result becomes
21        System.out.println(car.toString());
22
23    }
24
25 }
26
27
```

```
Console X
<terminated> Thirty [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (20 Aug 2024, 11:20:46 pm – 11:20:46 pm) [pid: 25480]
Suzuki
Ertiga
Blue
2020
Suzuki
Ertiga
Blue
2020
Suzuki
Ertiga
Blue
2020
```



```
1 package brocodejava;
2
3 public class Car {
4
5     String make = "Suzuki";
6     String model = "Ertiga";
7     int year = 2020;
8     String colour = "Blue";
9     double price = 500000.00;
10
11    public String toString() {
12        String myString = make + "\n" + model + "\n" + colour + "\n" + year;
13        return myString;
14    }
15
16    void drive() {
17        System.out.println("You are driving the car");
18    }
19
20    void brake() {
21        System.out.println("You have pressed the brake!");
22    }
23
24 }
25
```



```
Console ×
<terminated> Thirty [Java Application] C:\brocodejava.Car@4517d9a3
brocodejava.Car@4517d9a3
brocodejava.Car@4517d9a3
```

without `toString()` function in Car class

31. Array of Objects

The screenshot shows a Java development environment with two tabs: 'ThirtyOne.java' and 'Console'. The 'ThirtyOne.java' tab displays the following Java code:

```
1 package brocodejava;
2
3 public class ThirtyOne {
4
5     public static void main(String[] args) {
6
7         // Food[] refrigerator = new Food[3];
8
9         Food food1 = new Food("Pizza");
10        Food food2 = new Food("Burger");
11        Food food3 = new Food("Dosa");
12
13        // refrigerator[0] = food1;
14        // refrigerator[1] = food2;
15        // refrigerator[2] = food3;
16
17        Food[] refrigerator = {food1, food2, food3};
18
19        System.out.println(refrigerator[0].name);
20        System.out.println(refrigerator[1].name);
21        System.out.println(refrigerator[2].name);
22    }
23
24
25    class Food{
26        String name;
27        Food(String name){
28            this.name = name;
29        }
30    }
31}
```

The 'Console' tab shows the output of the program:

```
<terminated> ThirtyOne [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (20 A
Pizza
Burger
Dosa
```

32. Object Passing

The screenshot shows a Java application window with two tabs: 'ThirtyTwo.java' and 'Console'. The 'ThirtyTwo.java' tab displays the following Java code:

```
1 package brocodejava;
2
3 public class ThirtyTwo {
4
5     public static void main(String[] args) {
6
7         Garage garage = new Garage();
8         Cars car = new Cars("BMW");
9         Cars car2 = new Cars("Ertiga");
10
11        garage.park(car);
12        garage.park(car2);
13
14    }
15
16 }
17
18 class Cars{
19     String name;
20     Cars(String name){
21         this.name = name;
22     }
23 }
24
25 class Garage{
26
27     void park(Cars car) {
28         System.out.println("The "+car.name+" is parked in the garage");
29     }
30 }
```

The 'Console' tab shows the output of the program:

```
<terminated> ThirtyTwo [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (21 Aug 2024, 10:48:16 pm)
The BMW is parked in the garage
The Ertiga is parked in the garage
```

33. Static Keyword

The screenshot shows a Java IDE interface with two tabs: 'ThirtyThree.java' and 'Console'. The 'ThirtyThree.java' tab displays the following code:

```
1 package brocodejava;
2
3 public class ThirtyThree {
4
5     public static void main(String[] args) {
6         // static = modifier. A single copy of a variable/method is created and shared.
7         //           The class "owns" the static member
8
9         Friend friend1 = new Friend("Rajul");
10        Friend friend2 = new Friend("Sivaraman");
11        Friend friend3 = new Friend("Rayappan");
12        System.out.println(Friend.numberOrFriends);
13
14    }
15
16 }
17
18 class Friend{
19     String name;
20     static int numberOrFriends;
21     Friend(String name){
22         this.name = name;
23         numberOrFriends++;
24     }
25 }
```

The 'Console' tab shows the output of the program:

```
<terminated> ThirtyThree [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (21 Aug 2024, 11:29:42 pm – 11:29:42 pm)
3
```

if we remove the static keyword

The screenshot shows a Java IDE with a code editor displaying the following code:

```
System.out.println(Friend.numberOrFriends);
```

A tooltip is displayed over the line of code, stating:

Cannot make a static reference to the non-static field Friend.numberOrFriends

4 quick fixes available:

- Surround with try/catch
- Change 'numberOrFriends' to 'static'
- Create new instance of object 'PrintStream'
- Create reference to instance 'friend2' in 'Friend'

the class owns the singly copy of the static variable

hence every object which uses that class can access it and it will have the same value

The screenshot shows a Java IDE interface with two tabs: 'ThirtyThree.java' and 'Console'. The 'ThirtyThree.java' tab displays the following code:

```
1 package brocodejava;
2
3 public class ThirtyThree {
4
5     public static void main(String[] args) {
6         // static = modifier. A single copy of a variable/method is created and s
7         //           The class "owns" the static member
8
9         Friend friend1 = new Friend("Rajul");
10        Friend friend2 = new Friend("Sivaraman");
11        Friend friend3 = new Friend("Rayappan");
12        System.out.println(Friend.numberOrFriends);
13
14        Friend.displayFriends();
15    }
16
17}
18
19
20 class Friend{
21     String name;
22     static int numberOrFriends;
23     Friend(String name){
24         this.name = name;
25         numberOrFriends++;
26     }
27
28     static void displayFriends() {
29         System.out.println("You have "+numberOrFriends+" friends");
30     }
31 }
```

The 'Console' tab shows the output of running the program:

```
<terminated> ThirtyThree [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (21 Aug 2024, 11:37:35 pm)
3
You have 3 friends
```

34. Inheritance

The screenshot shows a Java code editor with a file named 'ThirtyFour.java' and a terminal window below it.

Code Editor (ThirtyFour.java):

```
1 package brocodejava;
2
3 public class ThirtyFour {
4
5     public static void main(String[] args) {
6         // Inheritance = the process where one class acquires the attributes and methods of another
7
8         Carz car = new Carz();
9         car.go();
10
11        Bicycle bike = new Bicycle();
12        bike.stop();
13
14        System.out.println(car.doors); //unique to the car class
15        System.out.println(bike.pedals); //unique to the bike class
16    }
17
18 }
19
20 class Vehicle{
21     double speed;
22
23     void go() {
24         System.out.println("This vehicle is moving");
25     }
26
27     void stop() {
28         System.out.println("This vehicle has stopped");
29     }
30 }
31
32 class Carz extends Vehicle{
33     int wheels = 4;
34     int doors = 4;
35 }
36
37 class Bicycle extends Vehicle{
38     int wheels = 2;
39     int pedals = 2;
40 }
41
42
43
```

Terminal (Console):

```
<terminated> ThirtyFour [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (21 Aug 2024, 11:44:02 pm – 11:44:02 pm) [pid: 274]
This vehicle is moving
This vehicle has stopped
4
2
```

35. Method Overriding

The screenshot shows a Java development environment with two tabs: 'ThirtyFive.java' and 'Console'.

ThirtyFive.java:

```
1 package brocodejava;
2
3 public class ThirtyFive {
4
5     public static void main(String[] args) {
6         // method overriding = declaring a method in a subclass
7         // which is already present in parent class
8         // done so that a child class can give its own implementation
9         Animal animal = new Animal();
10        animal.speak();
11
12        Dog dog = new Dog();
13        dog.speak();
14
15        Cat cat = new Cat();
16        cat.speak();
17    }
18
19 }
20
21 class Animal{
22     void speak() {
23         System.out.println("The animal is speaking");
24     }
25 }
26
27 class Cat extends Animal{
28 }
29
30 class Dog extends Animal{
31     @Override
32     void speak() {
33         System.out.println("The dog is barking");
34     }
35 }
```

Console:

```
<terminated> ThirtyFive [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 12:29:51 am – 12:29:51 am) [pid: 12]
The animal is speaking
The dog is barking
The animal is speaking
```

36. Super Keyword

```
ThirtySix.java ×

4
5  public static void main(String[] args) {
6      // super = keyword refers to the superclass (parent) of an object.
7      //           very similar to the "this" keyword.
8
9      Hero hero1 = new Hero("Batman",42,"$$$");
10     Hero hero2 = new Hero("Shaktimaan",75,"infinity");
11
12     System.out.println(hero1.name);
13     System.out.println(hero1.age);
14     System.out.println(hero1.power);
15     System.out.println();
16     System.out.println(hero2.toString());
17     System.out.println(hero1.toString());
18 }
19 }
20
21 class Person{
22     String name;
23     int age;
24
25     Person(String name,int age){
26         this.name = name;
27         this.age = age;
28     }
29
30     public String toString() { //why are we using public here?
31         return this.name + "\n" + this.age + "\n";
32     }
33
34 }
35
36 class Hero extends Person{
37     String power;
38     Hero(String name, int age, String power){
39         super(name,age);
40         this.power=power;
41     }
42
43     public String toString() { //why are we using public here?
44         return super.toString() + this.power + "\n";
45     }
46 }

Console ×
<terminated> ThirtySix [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 12:45:40 am -)
Batman
42
$$$

Shaktimaan
75
infinity

Batman
42
$$$
```

37. Abstraction

The screenshot shows a Java code editor and a terminal window. The code editor displays `ThirtySeven.java` with the following content:

```
1 package brocodejava;
2
3 public class ThirtySeven {
4
5     public static void main(String[] args) {
6         // abstract = abstract classes cannot be instantiated, but they can have a subclass
7         //           abstract methods are declared without an implementation
8         //           adding abstract keyword adds a level of security
9         //Vehicles vehicle = new Vehicles();
10        Carzz car = new Carzz();
11        car.go();
12    }
13}
14
15}
16
17 abstract class Vehicles{ // by using abstract keyword, vehicle cannot be declared abstractly and only the child objects can be created.
18     abstract void go(); //Abstract methods do not specify a body
19
20
21 }
22
23 class Carzz extends Vehicles{
24     @Override
25     void go() {
26         System.out.println("The driver is driving car");
27     }
28 }
```

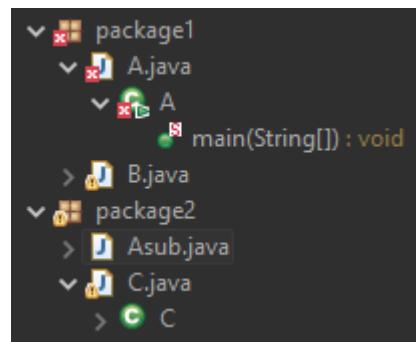
The terminal window below shows the output of the program:

```
<terminated> ThirtySeven [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 12:54:47 am – 12:54:47 am) [pid: 24332]
The driver is driving car
```

38. Access Modifiers

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
<i>no modifier</i>	Y	Y	N	N
private	Y	N	N	N



```

A.java   B.java   C.java X  Asub.java
1 package package2;
2 import package1.*;
3
4 public class C {
5     String defaultMessage = "This is the default";
6
7 }
8
9
10
  
```

```

A.java X  B.java   C.java  Asub.java
1 package package1;
2 import package2.*;
3
4 public class A {
5
6     public static void main(String[] args) {
7
8         C c = new C();
9         System.out.println(c.defaultMessage);
10    }
11
12 }
13
14
  
```

The field C.defaultMessage is not visible
 2 quick fixes available:
[Change visibility of 'defaultMessage' to 'public'](#)
[Create getter and setter for 'defaultMessage'...](#)

The screenshot shows a Java development environment with four tabs at the top: A.java, B.java, C.java, and Asub.java. The Asub.java tab is active, displaying the following code:

```
1 package package2;
2 import package1.*;
3
4 public class Asub extends A {
5
6     public static void main(String[] args) {
7
8         C c = new C();
9         System.out.println(c.defaultMessage);
10    }
11
12 }
13
```

Below the editor is a console window titled "Console X" showing the output of a terminated Java application named "Asub". The output text is:

```
<terminated> Asub [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22)
This is the default
```

using public keyword

The screenshot shows a Java IDE interface with two tabs open: A.java and C.java. The C.java tab is active, displaying the following code:

```
1 package package2;
2 import package1.*;
3
4 public class C {
5
6
7     public String publicMessage = "This is public";
8     String defaultMessage = "This is the default";
9
10 }
11
12
13
```

Below the editor is a terminal window titled "Console" showing the output of the program:

```
<terminated> A [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024)
This is public
```

The screenshot shows a Java IDE interface with two tabs open: A.java and C.java. The A.java tab is active, displaying the following code:

```
1 package package1;
2 import package2.*;
3
4 public class A {
5
6     public static void main(String[] args) {
7
8         C c = new C();
9         // System.out.println(c.defaultMessage);
10        System.out.println(c.publicMessage);
11
12    }
13
14 }
15
```

Below the editor is a terminal window titled "Console" showing the output of the program:

```
<terminated> A [Java Application] C:\Program Files\Java\jdk-22\bin\
This is public
```

If we remove the keyword `public` from the class `C`, the methods and attributes of that class wont be accessible to classes of other package

`protected` keyword

```
A.java × B.java × C.java × Asub.java ×
1 package package1;
2 import package2.*;
3
4 public class A {
5
6     protected String protectedMessage = "This is protected";
7
8
9 }
10
```

```
A.java × B.java × C.java × Asub.java ×
1 package package2;
2 import package1.*;
3
4 public class Asub extends A {
5
6     public static void main(String[] args) {
7
8         Asub asub = new Asub();
9         System.out.println(asub.protectedMessage);
10        // C c = new C();
11        // System.out.println(c.defaultMessage);
12        // System.out.println(c.publicMessage);
13
14    }
15
16 }
17
```

Console ×

```
<terminated> Asub [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (This is protected)
```

we are able to access protectedMessage from different package as Asub extends A and the A contains protected variable

```
A.java × B.java × C.java × *Asub.java ×
1 package package1;
2 import package2.*;
3
4 public class B {
5
6     private String privateMessage = "This is the private"; //something that is private is only visible to class that it contains itself!
7
8 }
9
```

A screenshot of an IDE showing Java code in A.java. The code includes imports for package2.* and package1.*. It defines class A with a protected String protectedMessage and a public static void main method. Inside main, it creates an Asub object and prints its protected message. It also creates a C object and prints its default and public messages. Below this, it creates a B object and prints its private message. A tooltip appears over the line System.out.println(b.privateMessage); stating "The field B.privateMessage is not visible". It offers two quick fixes: "Change visibility of 'privateMessage' to 'package'" and "Create getter and setter for 'privateMessage'...". The console output shows "This is protected", "This is public", and "This is private".

```
1 import package2.*;
2
3
4 public class A {
5
6     protected String protectedMessage = "This is protected";
7     public static void main(String[] args) {
8
9         // Asub asub = new Asub();
10        // System.out.println(asub.protectedMessage);
11        // C c = new C();
12        // System.out.println(c.defaultMessage);
13        // System.out.println(c.publicMessage);
14
15        B b = new B();
16        System.out.println(b.privateMessage);
17    }
18}
```

The field B.privateMessage is not visible
2 quick fixes available:
Change visibility of 'privateMessage' to 'package'
Create getter and setter for 'privateMessage'...

```
This is protected
This is public
This is private
```

extra:

A screenshot of an IDE showing Java code in C.java. The code defines class C with a public String publicMessage, a protected String protectedMessage (with a detailed comment about accessibility), a default String defaultMessage, and a private String privateMessage. The private message is underlined with a red squiggle. A tooltip provides a detailed explanation of access modifiers: "public" is accessible from anywhere; "protected" is accessible within the same package or by subclasses in different packages; "default" is accessible within the same package; and "private" is accessible only within the same class. The console output shows "This is protected" and "This is public".

```
1 package package2;
2 import package1.*;
3
4 public class C {
5
6     public String publicMessage = "This is public";
7     protected String protectedMessage = "This is protected"; //something that is protected is accessible to a
9                                //different class in a different package as long as that class is a sub class of
10                               //whatever class contains this protected member
11     String defaultMessage = "This is the default";
12     private String privateMessage = "This is the private";
13
14 }
```

This is protected
This is public

39. Encapsulation

```
ThirtyNine.java ×
1 package brocodejava;
2
3 public class ThirtyNine {
4
5     public static void main(String[] args) {
6         // encapsulation = attributes of a class will be hidden or private,
7         // can be accessed only through methods (getters and setters)
8         // You should make attributes private if you don't have a reason to make them public/protected
9
10        Carss car = new Carss("Ertiga", "Suzuku", 2020);
11
12        System.out.println(car.make);
13    }
14
15 }
16
17 class Carss{
18     private String make;
19     private String model;
20     private int year;
21
22     Carss(String ma 4 quick fixes available:
23         this.make =
24         this.model =
25         this.year =
26     }
27 }
```

The value of the field Carss.model is not used

- ✖ Remove 'model', keep assignments with side effects
- Fix 3 problems of same category in file
- Create getter and setter for 'model'...
- Add @SuppressWarnings 'unused' to 'model'
- Configure problem severity

Press 'F2' for focus

```
// can be accessed only through methods (getters and setters)
// You should make attributes private if you don't have a reas
9
10       Carss car = new Carss("Ertiga", "Suzuku", 2020);
11
12       System.out.println(car.getMake());
13       System.out.println(car.getModel());
14       System.out.println(car.getYear());
15
16       car.year = 2022;
17
18   }  The field Carss.year is not visible
19   }  2 quick fixes available:
20
21   }
22
23 class Carss{  Press 'F2' for focus
24     private String make;
25     private String model;
26     private int year;
27
28     Carss(String make, String model, int year){
29         this.make = make;
30         this.model = model;
31         this.year = year;
32     }
33
34     public String getMake() {
35         return make;
36     }
37
38 }
```

The field Carss.year is not visible

- Change visibility of 'year' to 'package'
- Create getter and setter for 'year'...

Press 'F2' for focus

Console ×

```
<terminated> ThirtyNine [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 7:08:39 pm – 7:08:40 pm)
```

Ertiga
Suzuku
2020

The screenshot shows a Java code editor and a terminal window. The code editor displays a file named `ThirtyNine.java` with the following content:

```
1 package brocodejava;
2
3 public class ThirtyNine {
4
5     public static void main(String[] args) {
6         // encapsulation = attributes of a class will be hidden or private,
7         // can be accessed only through methods (getters and setters)
8         // You should make attributes private if you don't have a reason to make them public/protected
9
10        Carss car = new Carss("Ertiga", "Suzuku", 2020);
11
12        // car.year = 2022;
13        car.setYear(2022);
14
15        System.out.println(car.getMake());
16        System.out.println(car.getModel());
17        System.out.println(car.getYear());
18    }
19}
20 class Carss{
21    private String make;
22    private String model;
23    private int year;
24
25    Carss(String make, String model, int year){
26        this.make = make;
27        this.setModel(model);
28        this.setYear(year);
29    }
30
31    public String getMake() {
32        return make;
33    }
34    public String getModel() {
35        return model;
36    }
37    public int getYear() {
38        return year;
39    }
40
41    public void setMake(String make) {
42        this.make = make;
43    }
44    public void setModel(String model) {
45        this.model = model;
46    }
47    public void setYear(int year) {
48        this.year = year;
49    }
50}
```

The terminal window below shows the output of the program:

```
<terminated> ThirtyNine [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 7:12:15 pm – 7:12:15 pm) [pid: 1728]
Ertiga
Suzuku
2022
```

Other Classes do not have direct access to the values.

Instead they can use getter and setter methods to access them

40. Copy Objects

The screenshot shows a Java IDE interface with two tabs: "Forty.java" and "Console".

Forty.java:

```
1 package brocodejava;
2
3 public class Forty {
4
5     public static void main(String[] args) {
6         //Using Carss from thirtynine
7
8         Carss car1 = new Carss("Suzuki","Ertiga",2020);
9         Carss car2 = new Carss("Santro","Xing",2008);
10
11        //if we want to copy the values from car1 to car2
12        car2 = car1;
13
14        System.out.println(car1);
15        System.out.println(car2);
16        System.out.println();
17        System.out.println(car1.getMake());
18        System.out.println(car1.getModel());
19        System.out.println(car1.getYear());
20        System.out.println();
21        System.out.println(car2.getMake());
22        System.out.println(car2.getModel());
23        System.out.println(car2.getYear());
24    }
25
26 }
27
```

Console:

```
<terminated> Forty [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 7:1)
brocodejava.Carss@4517d9a3
brocodejava.Carss@4517d9a3

Suzuki
Ertiga
2020

Suzuki
Ertiga
2020
```

both will be referring to the same object, ie,, they point to the same car

```
Forty.java × ThirtyNine.java
1 package brocodejava;
2
3 public class Forty {
4
5     public static void main(String[] args) {
6         //Using Carss from thirtynine
7
8         Carss car1 = new Carss("Suzuki","Ertiga",2020);
9         Carss car2 = new Carss("Santro","Xing",2008);
10
11        //if we want to copy the values from car1 to car2
12        car2 = car1;
13        car2.copy(car1);
14
15        System.out.println(car1);
16        System.out.println(car2);
17        System.out.println();
18        System.out.println(car1.getMake());
19        System.out.println(car1.getModel());
20        System.out.println(car1.getYear());
21        System.out.println();
22        System.out.println(car2.getMake());
23        System.out.println(car2.getModel());
24        System.out.println(car2.getYear());
25    }
26
27 }
28
```

```
Console ×
<terminated> Forty [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 7:43:21 pm – 7:43:21 pm)
brocodejava.Carss@4517d9a3
brocodejava.Carss@372f7a8d

Suzuki
Ertiga
2020

Suzuki
Ertiga
2020
```

```
49      this.year = year,
50  }
51
52  public void copy(Carss x) {
53      this.setMake(x.getMake());
54      this.setModel(x.getModel());
55      this.setYear(x.getYear());
56  }
57
58 }
```

Alternatively, if we want to assign the values of car1 directly to car2

```
Forty.java × ThirtyNine.java
1 package brocodejava;
2
3 public class Forty {
4
5     public static void main(String[] args) {
6         //Using Carss from thirtynine
7
8         Carss car1 = new Carss("Suzuki","Ertiga",2020);
9 //        Carss car2 = new Carss("Santro","Xing",2008);
10
11 //        //if we want to copy the values from car1 to car2
12 //        car2 = car1;
13 //        car2.copy(car1);
14
15     Carss car2 = new Carss(car1);
16
17     System.out.println(car1);
18     System.out.println(car2);
19     System.out.println();
20     System.out.println(car1.getMake());
21     System.out.println(car1.getModel());
22     System.out.println(car1.getYear());
23     System.out.println();
24     System.out.println(car2.getMake());
25     System.out.println(car2.getModel());
26     System.out.println(car2.getYear());
27 }
28
29 }
```

```
29         this.setYear(year);
30     }
31
32     //Overloaded constructors
33     Carss(Carss x){
34         this.copy(x);
35     }
36
```

41. Interfaces

```

1 *FortyOne.java × 2 Prey.java 3 Predator.java
1 package brocodejava;
2
3 public class FortyOne {
4
5     public static void main(String[] args) {
6         // interface = a template that can be applied to a class.
7         //           similar to inheritance, but specifies what a class has/must do.
8         //           difference between classes and interface = classes can apply more than one interface, inheritance is limited to 1 super class
9
10        Rabbit rabbit = new Rabbit();
11        rabbit.flee();
12
13        Hawk hawk = new Hawk();
14        hawk.hunt();
15
16        Fish fish = new Fish();
17        fish.hunt();
18        fish.flee();
19    }
20    class Rabbit implements Prey{
21
22        public void flee() {
23            System.out.println("The rabbit is fleeing");
24        }
25    }
26
27    class Hawk implements Predator{
28
29        public void hunt() {
30            System.out.println("The Hawk is hunting");
31        }
32    }
33
34
35    class Fish implements Prey,Predator{
36
37        public void hunt() {
38            System.out.println("The Fish is hunting");
39        }
40        public void flee() {
41            System.out.println("The Fish is fleeing");
42        }
43    }

```

Console

```

<terminated> FortyOne [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 8:17:27 pm - 8:17:27 pm) [pid: 20696]
The rabbit is fleeing
The Hawk is hunting
The Fish is hunting
The Fish is fleeing

```

```

1 *FortyOne.java × 2 Prey.java × 3 Predator.java
1 package brocodejava;
2
3 public interface Prey {
4     void flee(); //do not need body
5 }
6

```

```

1 *FortyOne.java 2 Prey.java 3 Predator.java ×
1 package brocodejava;
2
3 public interface Predator {
4     void hunt();
5 }
6

```

The screenshot shows a Java IDE interface with two tabs open: 'FortyOne.java' and 'Prey.java'. The 'FortyOne.java' tab contains the following code:

```
1 package brocodejava;
2
3 public class FortyOne {
4
5     public static void main(String[] args) {
6         // interface = a template that can be applied to a class.
7         //           similar to inheritance, but specifies what a class has/must do.
8         //           difference between classes and interface = classes can apply more than one interface, inheritance is limited to 1 super class
9
10        Rabbit rabbit = new Rabbit();
11        rabbit.flee();
12
13        Hawk hawk = new Hawk();
14        hawk.hunt();
15
16        Fish fish = new Fish();
17        fish.hunt();
18        fish.flee();
19    }
20 }
21 class Rabbit implements Prey{
22
23     public void flee() {
24         System.out.println("The rabbit is fleeing");
25     }
26 }
27 class Hawk implements Predator{
28
29     public void hunt() {
30         System.out.println("The Hawk is hunting");
31     }
32 }
33
34 }
35 class Fish implements Prey,Predator{
36
37     public void hunt() {
38         System.out.println("The Fish is hunting");
39     }
40     public void flee() {
41         System.out.println("The Fish is fleeing");
42     }
43 }
```

The 'Prey.java' tab is visible in the background. Below the code editor is a terminal window titled 'Console' showing the output of the application:

```
<terminated> FortyOne [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 8:17:27 pm – 8:17:27 pm) [pid: 20696]
The rabbit is fleeing
The Hawk is hunting
The Fish is hunting
The Fish is fleeing
```

42. Polymorphism

The screenshot shows a Java code editor and a terminal window. The code editor displays `FortyTwo.java` with syntax highlighting for Java keywords and comments. The terminal window below shows the execution of the program and its output.

```
1 package brocodejava;
2
3 public class FortyTwo {
4
5     public static void main(String[] args) {
6         // polymorphism = The ability of an object to identify as more than one type
7
8         CARO car = new CARO();
9         BICYCLEO bicycle = new BICYCLEO();
10        BOATO boat = new BOATO();
11
12        // Can store all types of vehicles (car, bicycle, boat) in the same array
13        VEHICLEO[] racers = {car, bicycle, boat};
14
15        car.go();
16        bicycle.go();
17        boat.go();
18
19        for (VEHICLEO x : racers) {
20            x.go();
21        }
22    }
23 }
24 class VEHICLEO {
25     public void go() {
26         // Default behavior for vehicles
27     }
28 }
29
30 class BICYCLEO extends VEHICLEO {
31     @Override
32     public void go() {
33         System.out.println("The bicycle begins moving");
34     }
35 }
36 class CARO extends VEHICLEO {
37     @Override
38     public void go() {
39         System.out.println("The car begins moving");
40     }
41 }
42 class BOATO extends VEHICLEO {
43     @Override
44     public void go() {
45         System.out.println("The boat begins moving");
46     }
47 }
```

Console Output:

```
<terminated> FortyTwo [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (22 Aug 2024, 9:02:49 pm – 9:02:49 pm) [pi]
The car begins moving
The bicycle begins moving
The boat begins moving
The car begins moving
The bicycle begins moving
The boat begins moving
```

43. Dynamic Polymorphism

The screenshot shows a Java IDE interface with two tabs open: 'FortyThree.java' and 'ThirtyFive.java'. The code in 'FortyThree.java' demonstrates polymorphism through a 'main' method that creates objects of type 'Dog', 'Cat', or 'Animal' based on user input ('dog', 'cat', or anything else). The 'Animal' class has a 'speak()' method. The 'Cat' class overrides it to print 'meow'. The 'Dog' class does not override it, so it prints 'woof'. The 'Animal' class also overrides the equals() method to check for equality based on value, not memory address.

```
1 package brocodejava;
2 import java.util.*;
3 public class FortyThree {
4
5     public static void main(String[] args) {
6         //polymorphism = many forms
7         // dynamic = after compilation, during runtime
8         Scanner scanner = new Scanner(System.in);
9         Animal animal;
10
11        System.out.println("What animal do u want?");
12        String choice = scanner.next();
13        System.out.println(choice);
14        if (choice.equals("dog")) { //using == will check for the memory values also hence would return false
15            animal = new Dog();
16            animal.speak();
17        }
18        else if(choice.equals("cat")) { //therefore using .equals() is suitable in this case as it checks for the values only
19            animal = new Cat();
20            animal.speak();
21        }
22        else {
23            animal = new Animal();
24            animal.speak();
25        }
26    }
27 }
28 }
```

The 'Console' tab shows the application's output:

```
<terminated> FortyThree [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 11:48:07 am – 11:48:42 am) [pid: 19832]
What animal do u want?
cat
cat
The cat says meow
```

44. Exception Handling

The screenshot shows a Java IDE interface with two tabs open: 'FortyFour.java' and 'ThirtyFive.java'. The code in 'FortyFour.java' attempts to divide two integers entered by the user. It uses a try-catch block to handle a potential 'ArithmaticException' that occurs if the divisor is zero. The code prompts the user for two numbers, performs the division, and prints the result. However, if the divisor is zero, it catches the exception and prints an error message.

```
1 package brocodejava;
2 import java.util.Scanner;
3 public class FortyFour {
4
5     public static void main(String[] args) {
6         // exception = an event that occurs during the execution of a program that disrupts the normal flow of instructions
7         Scanner scanner = new Scanner(System.in);
8
9         System.out.println("Enter a whole number to divide");
10        int x = scanner.nextInt();
11
12        System.out.println("Enter a whole number to divide by: ");
13        int y = scanner.nextInt();
14
15        int z = x/y;
16        System.out.println("result: "+z );
17    }
18 }
19
20 }
```

The 'Console' tab shows the application's output:

```
<terminated> FortyFour [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 11:51:34 am – 11:51:37 am) [pid: 19756]
Enter a whole number to divide
5
Enter a whole number to divide by:
0
Exception in thread "main" java.lang.ArithmaticException: / by zero
        at brocodejava.FortyFour.main(FortyFour.java:15)
```

The screenshot shows a Java development environment with two panes. The top pane is titled "FortyFour.java" and contains the following Java code:

```
1 package brocodejava;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4 public class FortyFour {
5
6     public static void main(String[] args) {
7         // exception = an event that occurs during the execution of a program that disrupts the normal flow of instructions
8         Scanner scanner = new Scanner(System.in);
9         try {
10             System.out.println("Enter a whole number to divide");
11             int x = scanner.nextInt();
12
13             System.out.println("Enter a whole number to divide by: ");
14             int y = scanner.nextInt();
15
16             int z = x/y;
17             System.out.println("result: "+z );
18         }
19         catch(ArithmaticException e) {
20             System.out.println("You cannot divide by 0 rahul");
21         }
22         catch(InputMismatchException e) {
23             System.out.println("Enter number da tpmf");
24         }
25         catch(Exception e) {
26             System.out.println("Something went wrong");
27         }
28         finally {
29             System.out.println("This will always execute");
30             scanner.close();
31         }
32     }
33 }
```

The bottom pane is titled "Console" and shows the terminal output of the application:

```
<terminated> FortyFour [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 12:29:49 pm – 12:29:51 pm) [pid: 30764]
Enter a whole number to divide
10
Enter a whole number to divide by:
0
You cannot divide by 0 rahul
This will always execute
```

```

FortyFour.java ×
1 package brocodejava;
2 import java.util.InputMismatchException;
3 import java.util.Scanner;
4 public class FortyFour {
5
6     public static void main(String[] args) {
7         // exception = an event that occurs during the execution of a program that disrupts the normal flow of instructions
8         Scanner scanner = new Scanner(System.in);
9         try {
10             System.out.println("Enter a whole number to divide");
11             int x = scanner.nextInt();
12
13             System.out.println("Enter a whole number to divide by: ");
14             int y = scanner.nextInt();
15
16             int z = x/y;
17             System.out.println("result: "+z );
18         }
19         catch(ArithmaticException e) {
20             System.out.println("You cannot divide by 0 rahul");
21         }
22         catch(InputMismatchException e) {
23             System.out.println("Enter number da tpmf");
24         }
25         catch(Exception e) {
26             System.out.println("Something went wrong");
27         }
28         finally {
29             System.out.println("This will always execute");
30             scanner.close();
31         }
32     }
33 }

```

Console ×

```

<terminated> FortyFour [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 12:30:16 pm – 12:30:19 pm) [pid: 30676]
Enter a whole number to divide
10
Enter a whole number to divide by:
pizza
Enter number da tpmf
This will always execute

```

45. File Class

```

FortyFive.java ×
1 package brocodejava;
2 import java.io.File;
3 public class FortyFive {
4
5     public static void main(String[] args) {
6         // file = abstract representation of file and directory pathnames
7
8         File file = new File("C:\\Users\\visha\\eclipse-workspace\\brocodejava\\src\\brocodejava\\secret_message.txt");
9         if (file.exists()) {
10             System.out.println("That file exists");
11             System.out.println(file.getPath());
12             System.out.println(file.getAbsolutePath());
13             System.out.println(file.isFile());
14             //file.delete();
15         }
16         else {
17             System.out.println("That file doesn't exist");
18         }
19
20     }
21
22 }
23

```

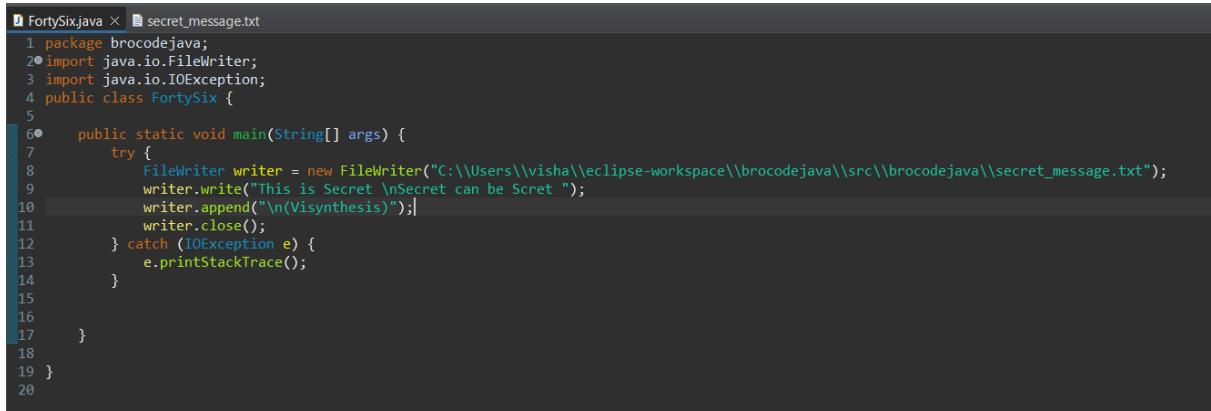
Console ×

```

<terminated> FortyFive [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 12:59:00 pm – 12:59:00 pm) [pid: 12032]
That file exists
C:\Users\visha\eclipse-workspace\brocodejava\src\brocodejava\secret_message.txt
C:\Users\visha\eclipse-workspace\brocodejava\src\brocodejava\secret_message.txt
true

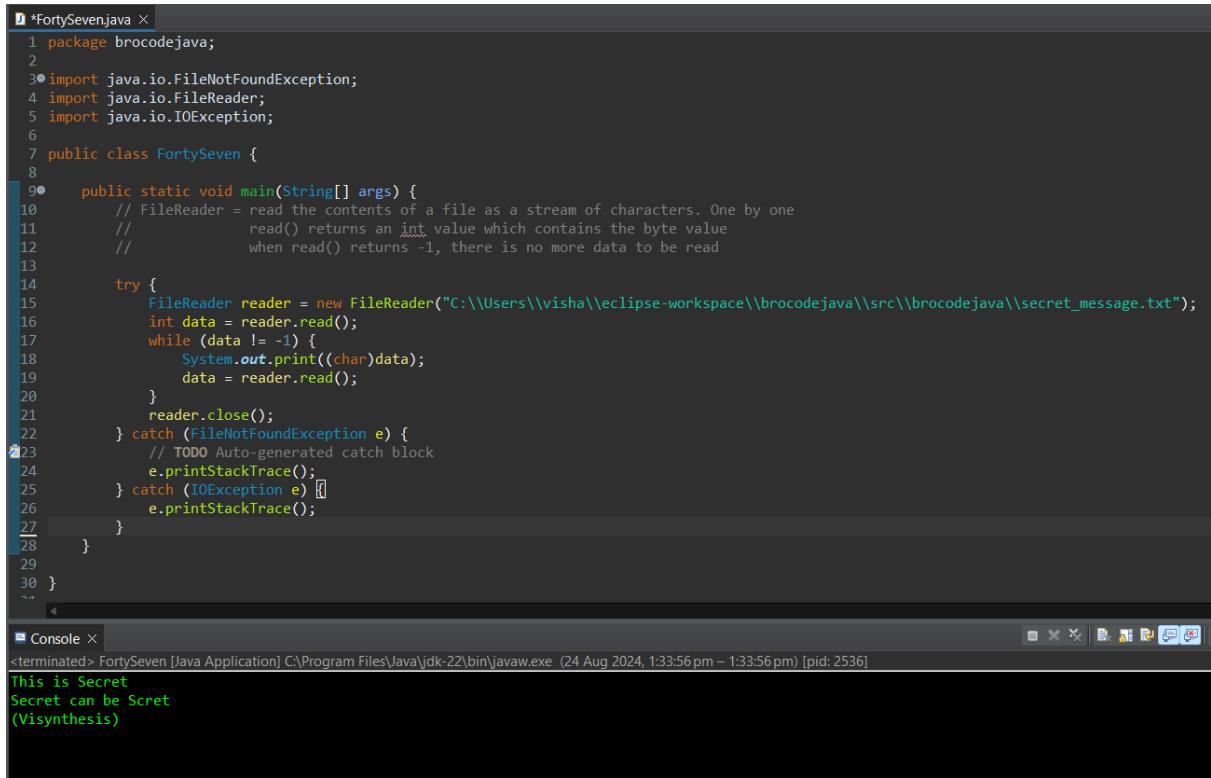
```

46. File Writer



```
FortySix.java × secret_message.txt
1 package brocodejava;
2 import java.io.FileWriter;
3 import java.io.IOException;
4 public class FortySix {
5
6     public static void main(String[] args) {
7         try {
8             FileWriter writer = new FileWriter("C:\\\\Users\\\\visha\\\\eclipse-workspace\\\\brocodejava\\\\src\\\\brocodejava\\\\secret_message.txt");
9             writer.write("This is Secret \nSecret can be Scret ");
10            writer.append("\nVisynthesis");
11            writer.close();
12        } catch (IOException e) {
13            e.printStackTrace();
14        }
15    }
16}
17}
18}
19}
20}
```

47. File Reader



```
*FortySeven.java ×
1 package brocodejava;
2
3 import java.io.FileNotFoundException;
4 import java.io.FileReader;
5 import java.io.IOException;
6
7 public class FortySeven {
8
9     public static void main(String[] args) {
10         // FileReader = read the contents of a file as a stream of characters. One by one
11         //           read() returns an int value which contains the byte value
12         //           when read() returns -1, there is no more data to be read
13
14         try {
15             FileReader reader = new FileReader("C:\\\\Users\\\\visha\\\\eclipse-workspace\\\\brocodejava\\\\src\\\\brocodejava\\\\secret_message.txt");
16             int data = reader.read();
17             while (data != -1) {
18                 System.out.print((char)data);
19                 data = reader.read();
20             }
21             reader.close();
22         } catch (FileNotFoundException e) {
23             // TODO Auto-generated catch block
24             e.printStackTrace();
25         } catch (IOException e) {
26             e.printStackTrace();
27         }
28     }
29 }
30 }
```

Console X

```
<terminated> FortySeven [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 1:33:56 pm – 1:33:56 pm) [pid: 2536]
This is Secret
Secret can be Scret
(Visynthesis)
```

48-76 SKIPPED FOR NOW

TO visit: Serialization

77. Threads

In Java, the `InterruptedException` is a checked exception that is thrown when a thread that is sleeping, waiting, or otherwise occupied is interrupted. It is used to handle situations where a thread is waiting or sleeping and another thread wants to interrupt it and make it stop waiting or sleeping.

Common Scenarios Where `InterruptedException` Is Thrown:

1. **Sleeping:** When a thread is put to sleep using `Thread.sleep()`, it can be interrupted by another thread. This will throw an `InterruptedException`.

```
java Copy code  
  
try {  
    Thread.sleep(1000); // Sleep for 1 second  
} catch (InterruptedException e) {  
    // Handle the interruption  
    System.out.println("Thread was interrupted");  
}
```

2. **Waiting:** When a thread is waiting on an object using methods like `Object.wait()` or `Thread.join()`, it can be interrupted.

```
java Copy code  
  
synchronized (someObject) {  
    try {  
        someObject.wait(); // Wait until notified  
    } catch (InterruptedException e) {  
        // Handle the interruption  
        System.out.println("Thread was interrupted while waiting");  
    }  
}
```

3. **Joining:** When a thread is waiting for another thread to finish using `Thread.join()`, it can be interrupted.

```
java Copy code
      Thread t = new Thread(() -> {
          // Thread logic
      });
      t.start();
      try {
          t.join(); // Wait for t to finish
      } catch (InterruptedException e) {
          // Handle the interruption
          System.out.println("Thread was interrupted while waiting for another thread to fin
      }
```

Why Use `throws InterruptedException`:

- **Declaration Requirement:** If your method calls a method that can throw `InterruptedException` (like `Thread.sleep()`), you either need to handle the exception with a try-catch block or declare that your method can throw it using `throws InterruptedException`.
- **Propagating the Exception:** By using `throws InterruptedException`, you allow the calling method to decide how to handle the interruption. This is useful if your method is part of a larger thread management system.

```
java Copy code
      public void someMethod() throws InterruptedException {
          Thread.sleep(1000); // Could throw InterruptedException
      }
```

Thread = a thread of execution in a program (kind of like a virtual cpu)
A JVM allows an application to have multiple threads running concurrently
Each thread can execute parts of your code in parallel with the main thread
Each thread has a priority
Threads with higher priority are executed in preference compared to threads with lower priority

The Java Virtual Machine continues to execute threads until either of the following occurs

1. The exit method of class Runtime has been called
2. All user threads have died

When a JVM starts up, there is a thread which calls the main method

This thread is called "main"

Daemon thread is a low priority thread that runs in background to perform tasks such as garbage collection

JVM terminates itself when all user threads (non-daemon threads) finish their execution

The screenshot shows a Java IDE interface with two tabs: 'SeventySeven.java' and 'Console'. The code in 'SeventySeven.java' is as follows:

```
1 package brocodejava;
2
3 public class SeventySeven {
4
5     public static void main(String[] args) throws InterruptedException {
6
7         System.out.println(Thread.activeCount());
8         Thread.currentThread().setName("Z3R3F");
9         System.out.println(Thread.currentThread().getName());
10        Thread.currentThread().setPriority(10); //Highest Priority
11        System.out.println(Thread.currentThread().getPriority()); //Higher the number, higher the priority
12
13        //to check if the current thread is alive
14        System.out.println(Thread.currentThread().isAlive());
15
16        for (int i=3;i>0;i--) {
17            System.out.println(i);
18            Thread.sleep(1000);
19        }
20        System.out.println("You are done!");
21    }
22
23 }
```

The 'Console' tab shows the output of the program:

```
1
Z3R3F
10
true
3
2
1
You are done!
```

```
*SeventySeven.java × MyThread.java ×
3 public class SeventySeven {
4
5•     public static void main(String[] args) throws InterruptedException {
6
7     System.out.println(Thread.activeCount());
8     Thread.currentThread().setName("Z3R3F");
9     System.out.println(Thread.currentThread().getName());
10    Thread.currentThread().setPriority(10); //Highest Priority
11    System.out.println(Thread.currentThread().getPriority()); //Higher the number, higher the priority
12    //to check if the current thread is alive
13    System.out.println(Thread.currentThread().isAlive());
14    for (int i=3;i>0;i--) {
15        System.out.println(i);
16        Thread.sleep(1000);
17    }
18    System.out.println("You are done!");
19
20    MyThread thread2 = new MyThread();
21
22    System.out.println("DAEMON?");
23    System.out.println(thread2.isDaemon());
24    System.out.println("DAEMON?");
25    thread2.setDaemon(true);
26    System.out.println(thread2.isDaemon());
27
28    thread2.start();
29    System.out.println(thread2.isAlive());
30    thread2.setName("Threadex");
31    System.out.println(thread2.getName());
32
33    System.out.println(thread2.getPriority()); //inherits the priority of the main thread
34    thread2.setPriority(9);
35    System.out.println(thread2.getPriority());
36
37    System.out.println(Thread.activeCount());
38 }
```

```
eventySeven.java × MyThread.java ×
package brocodejava;

public class MyThread extends Thread {

    //overriding the method run in thread class
    @Override
    public void run() {
        if(this.isDaemon()) {
            System.out.println("This is a Daemon thread that is running");
        }
        else {
            System.out.println("This is a user thread that is running");
        }
        System.out.println("This Thread is Running");
    }

}
```

```
<terminated> SeventySeven [Java Application] C:\Program  
1  
Z3R3F  
10  
true  
3  
2  
1  
You are done!  
DAEMON?  
false  
DAEMON?  
true  
true  
Threadex  
10  
9  
This is a Daemon thread that is running  
This Thread is Running  
2
```

78. Multi-Threading

```
SeventyEight.java × MyThread2.java × MyRunnable.java  
1 package brocodejava;  
2  
3 public class SeventyEight {  
4  
5     public static void main(String[] args) {  
6         // multithreading = process of executing multiple threads simultaneously  
7         // Helps maximum utilization of CPU  
8         // Threads are independent, they don't affect the execution of other threads  
9         // An exception in one thread will not interrupt other threads  
10        // useful for serving multiple clients, multiplayer games, or other mutually independent tasks  
11  
12        //1  
13        //create a subclass of the thread class, create a class and make sure it extends thread class,  
14        //and then you have access to the run method  
15        MyThread2 thread1 = new MyThread2();  
16  
17        //2  
18        //other way to create a thread -> using subclass or class thats implementing runnable interface,  
19        //create instance and pass it as an argument to the thread class  
20        MyRunnable runnable1 = new MyRunnable();  
21        Thread thread2 = new Thread(runnable1);  
22  
23        thread1.start();  
24        thread2.start();|
```

SeventyEight.java × MyThread2.java × MyRunnable.java

```
1 package brocodejava;
2
3 public class MyThread2 extends Thread {
4
5     public void run() {
6
7         for(int i=10;i>0;i--) {
8             System.out.println("Thread #1 : "+i);
9             try {
10                 Thread.sleep(1000);
11             } catch (InterruptedException e) {
12                 // TODO Auto-generated catch block
13                 e.printStackTrace();
14             }
15         }
16         System.out.println("Thread #1 is finished");
17     }
18 }
19
```

SeventyEight.java × MyThread2.java × MyRunnable.java

```
1 package brocodejava;
2
3 public class MyRunnable implements Runnable{
4     public void run() {
5
6         for(int i=0;i<10;i++) {
7             System.out.println("Thread #2 : "+i);
8             try {
9                 Thread.sleep(1000);
10            } catch (InterruptedException e) {
11                // TODO Auto-generated catch block
12                e.printStackTrace();
13            }
14        }
15        System.out.println("Thread #2 is finished");
16    }
17 }
18
19 }
```

```
Console ×
<terminated> SeventyEight [Java Application] C:\Pr
Thread #1 : 10
Thread #2 : 0
Thread #2 : 1
Thread #1 : 9
Thread #1 : 8
Thread #2 : 2
Thread #1 : 7
Thread #2 : 3
Thread #1 : 6
Thread #2 : 4
Thread #1 : 5
Thread #2 : 5
Thread #1 : 4
Thread #2 : 6
Thread #1 : 3
Thread #2 : 7
Thread #1 : 2
Thread #2 : 8
Thread #1 : 1
Thread #2 : 9
Thread #1 is finished
Thread #2 is finished
```

Intentionally causing one thread to stop executing

The screenshot shows a Java development environment with two tabs open: "MyThread2.java" and "MyRunnable.java". The "MyThread2.java" tab is active, displaying the following code:

```
1 package brocodejava;
2
3 public class MyThread2 extends Thread {
4
5     public void run() {
6
7         for(int i=10;i>0;i--) {
8             System.out.println("Thread #1 : "+i);
9             try {
10                 Thread.sleep(1000);
11             } catch (InterruptedException e) {
12                 // TODO Auto-generated catch block
13                 e.printStackTrace();
14             }
15             System.out.println(i/0);
16         }
17         System.out.println("Thread #1 is finished");
18     }
19 }
```

The "Console" tab below shows the execution output:

```
<terminated> SeventyEight [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024)
Thread #1 : 10
Thread #2 : 0
Thread #2 : 1
Exception in thread "Thread-0" java.lang.ArithmetricException: / by zero
        at brocodejava.MyThread2.run(MyThread2.java:15)
Thread #2 : 2
Thread #2 : 3
Thread #2 : 4
Thread #2 : 5
Thread #2 : 6
Thread #2 : 7
Thread #2 : 8
Thread #2 : 9
Thread #2 is finished
```

```
SeventyEight.java X MyThread2.java MyRunnable.java
12     //1
13     //create a subclass of the thread class, create a class and make sure it extends thread class
14     //and then you have access to the run method
15     MyThread2 thread1 = new MyThread2();
16
17     //2
18     //other way to create a thread -> using subclass or class thats implementing runnable interface
19     //create instance and pass it as an argument to the thread class
20     MyRunnable runnable1 = new MyRunnable();
21     Thread thread2 = new Thread(runnable1);
22
23     thread1.start();
24     thread2.start();
25     System.out.println([1/0]);
26
27 }
28
29 }
30
```

Console X

```
SeventyEight [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 7:59:18 pm) [pid: 13768]
thread #1 : 10
Exception in thread "main" Thread #2 : 0
java.lang.ArithmetricException: / by zero
    at brocodejava.SeventyEight.main(SeventyEight.java:25)
thread #1 : 9
thread #2 : 1
thread #1 : 8
thread #2 : 2
thread #2 : 3
thread #1 : 7
```

Using join()

```
SeventyEight.java X MyThread2.java MyRunnable.java
1 package brocodejava;
2
3 public class SeventyEight {
4
5•     public static void main(String[] args) throws InterruptedException {
6         // multithreading = process of executing multiple threads simultaneously
7         // Helps maximum utilization of CPU
8         // Threads are independent, they don't affect the execution of other threads
9         // An exception in one thread will not interrupt other threads
10        // useful for serving multiple clients, multiplayer games, or other mutually independent tasks
11
12        //1
13        //create a subclass of the thread class, create a class and make sure it extends thread class,
14        //and then you have access to the run method
15        MyThread2 thread1 = new MyThread2();
16
17        //2
18        //other way to create a thread -> using subclass or class thats implementing runnable interface,
19        //create instance and pass it as an argument to the thread class
20        MyRunnable runnable1 = new MyRunnable();
21        Thread thread2 = new Thread(runnable1);
22
23        thread1.start();
24    //    thread1.join(); //Main thread is going to wait until thread1 is finished and then execute thread2
25    thread1.join(3000); //Main thread will be paused for 3s before it will continue
26        thread2.start();
27    //    System.out.println(1/0);
28

Console X
<terminated> SeventyEight [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 8:02:06 pm – 8:02:19 pm) [pid: 19156]
Thread #1 : 10
Thread #1 : 9
Thread #1 : 8
Thread #2 : 0
Thread #1 : 7
Thread #2 : 1
Thread #1 : 6
Thread #2 : 2
Thread #1 : 5
Thread #2 : 3
Thread #1 : 4
Thread #2 : 4
```

The screenshot shows an IDE interface with two tabs open: 'SeventyEight.java' and 'MyThread2.java'. The code in 'SeventyEight.java' demonstrates creating threads using both the Thread class and the Runnable interface. It includes comments explaining the use of multithreading and the join() method. The 'Console' tab shows the output of the program, which prints numbers from 1 to 10 on separate lines, followed by 'Thread #1 is finished' and then the same sequence of numbers again, indicating that both threads have completed their execution.

```

1 public static void main(String[] args) throws InterruptedException {
2     // multithreading = process of executing multiple threads simultaneously
3     // Helps maximum utilization of CPU
4     // Threads are independent, they don't affect the execution of other threads
5     // An exception in one thread will not interrupt other threads
6     // useful for serving multiple clients, multiplayer games, or other mutually independent tasks
7
8     //1
9     //create a subclass of the thread class, create a class and make sure it extends thread class,
10    //and then you have access to the run method
11    MyThread2 thread1 = new MyThread2();
12
13    //2
14    //other way to create a thread -> using subclass or class that's implementing runnable interface,
15    //create instance and pass it as an argument to the thread class
16    MyRunnable runnable1 = new MyRunnable();
17    Thread thread2 = new Thread(runnable1);
18
19    thread1.start();
20    thread1.join(); //Main thread is going to wait until thread1 is finished and then execute thread2
21    // thread1.join(3000); //Main thread will be paused for 3s before it will continue
22    thread2.start();
23    // System.out.println(1/0);
24
25    }
26
27    }
28
29    }
30
31    }
32

```

Console output:

```

Thread #1 : 6
Thread #1 : 5
Thread #1 : 4
Thread #1 : 3
Thread #1 : 2
Thread #1 : 1
Thread #1 is finished
Thread #2 : 0
Thread #2 : 1
Thread #2 : 2
Thread #2 : 3

```

Daemon thread

The screenshot shows an IDE interface with three tabs open: 'SeventyEight.java', 'MyThread2.java', and 'MyRunnable.java'. The code in 'SeventyEight.java' demonstrates creating a daemon thread by setting the setDaemon(true) method. The 'Console' tab shows the output of the program, which ends with an ArithmeticException at line 30, indicating that the main thread has terminated while waiting for the daemon thread to finish.

```

17    //2
18    //other way to create a thread -> using subclass or class that's implementing runnable interface,
19    //create instance and pass it as an argument to the thread class
20    MyRunnable runnable1 = new MyRunnable();
21    Thread thread2 = new Thread(runnable1);
22
23    //Daemon thread = background thread (non-user) JVM will not wait for daemon thread to finish
24    thread1.setDaemon(true);
25    thread2.setDaemon(true);
26    thread1.start();
27    // thread1.join(); //Main thread is going to wait until thread1 is finished and then execute thread2
28    // thread1.join(3000); //Main thread will be paused for 3s before it will continue
29    thread2.start();
30    System.out.println(1/0);
31
32    }
33
34    }
35

```

Console output:

```

<terminated> SeventyEight [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe (24 Aug 2024, 8:04:56 pm – 8:04:56 pm) [pid: 2328]
Exception in thread "main" Thread #1 : 10
Thread #2 : 0
java.lang.ArithmetricException: / by zero
        at brocodejava.SeventyEight.main(SeventyEight.java:30)

```

79. Packages

The screenshot shows a Java code editor with two tabs open: "Main.java" and "Toolbox.java". The "Main.java" tab is active, displaying the following code:

```
1 import Tools.Toolbox;
2 import javax.swing.Icon;
3 public class Main {
4     public static void main(String[] args) {
5         Toolbox toolbox = new Toolbox();
6     }
7 }
8
9
```

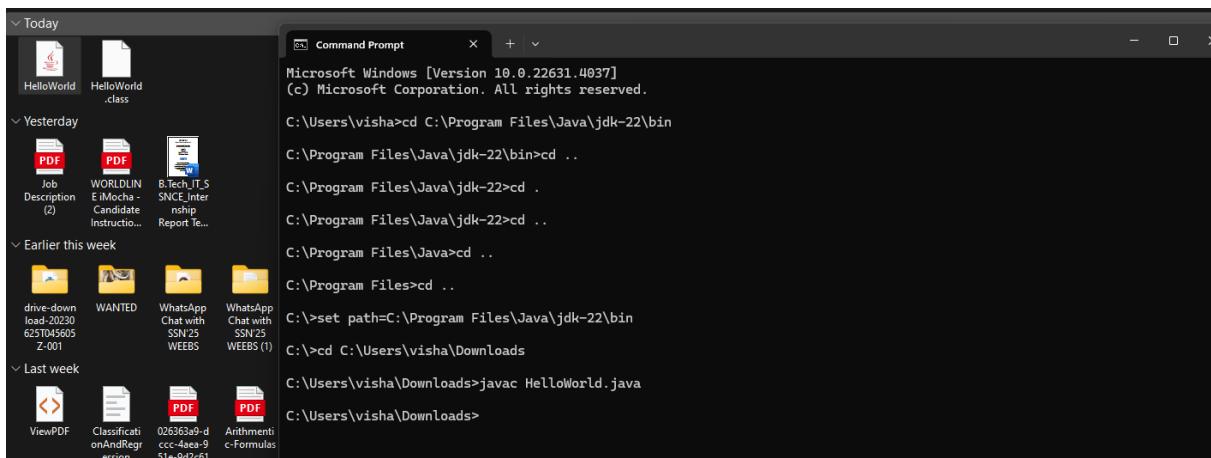
The screenshot shows a Java code editor with two tabs open: "Main.java" and "Toolbox.java". The "Toolbox.java" tab is active, displaying the following code:

```
1 package Tools;
2
3 public class Toolbox {
4
5 }
6
```

package name must be mentioned in any class except main

30. Command prompt

```
compile and run Java with Command Prompt
-----
1. Make sure you have a Java JDK installed (we did this in lesson #1)
2. (optional) use a text editor and save a file as .java
3. Open Command Prompt (windows) or Terminal (Mac) on your computer
4. set path=C:\Program Files\Java\jdk-13.0.1\bin (where JDK is located)
5. cd C:\Users\Cakow\Desktop (or wherever you java file is)
6. javac HelloWorld.java (to compile)
7. java HelloWorld (to run a .class file, it's portable)
```



```

C:\Users\visha\Downloads>java HelloWorld
HELLO WORLD

C:\Users\visha\Downloads>

```

31. executable (.jar file)

Create an executable jar with Eclipse IDE

1. Right click on Java project (pick a project with a GUI)
2. Export
3. Java > Runnable Jar file
4. At Launch configuration select your project
5. At Export destination, select where you want this jar file exported to
6. Finish

Create an executable jar with IntelliJ

1. File > Project Structure > Artifacts > (+) > JAR > jar from module with dependencies
2. Main Class: select the class containing your main method
3. OK

