Java Programs for JBL Lab

Tayyabali Sayyad

July 2022

1 Java Fundamentals

- Overview of procedure and object oriented Programming
- Java Designing Goals and Features of Java Language.
- Introduction to the principles of object-oriented programming:
- Classes
- Objects
- Abstraction
- Encapsulation Inheritance Polymorphism.
- Keywords, Data types, Variables, Operators,Expressions,
- Types of variables and methods.
- Control Statements: If Statement, If-else, Nested if,switch Statement, break, continue.
- Iteration Statements: for loop, while loop, and do-while loop

1.0.1 Simple Java program

```
public class Example {

    public static void main(String[] args) {
        System.out.println("This is a simple Java program.");
    }
}
```

1.1 Simple Java program 2

```
public class Example2 {

    public static void main(String[] args) {
        int num;

        num = 100;

        System.out.println("This is num: " + num);

        num = num * 2;

        System.out.print("The value of num * 2 is ");
        System.out.println(num);
    }
}
```

1.2 Demonstrate the if

```
public class IfSample {

    public static void main(String[] args) {
        int x, y;

        x = 10;
        y = 20;

        if (x < y)
            System.out.println("x is less than y");

        x = x * 2;

        if (x == y)
            System.out.println("x now equal to y");

        x = x * 2;
        if (x > y)
            System.out.println("x now greater than y");

        if (x == y)
            System.out.println("you won't see this");
    }
}
```

1.3 Demonstrate the for loop

```
public class ForTest {

    public static void main(String[] args) {
        int x;

        for (x = 0; x < 10; x = x + 1)
            System.out.println("This is x: " + x);
    }
}
```

1.4 Blocks in Java

```
public class BlockTest {

    public static void main(String[] args) {
        int x, y;

        y = 20;

        for (x = 0; x < 10; x++) {
            System.out.println("This is x: " + x);
            System.out.println("This is y: " + y);
            y = y - 2;
        }
    }
}
```

1.5 Compute distance light travels using long variables

```
public class Light {

    public static void main(String[] args) {
        int lightspeed;
        long days;
        long seconds;
        long distance;

        lightspeed = 186000;

        days = 1000;

        seconds = days * 24 * 60 * 60;

        distance = lightspeed * seconds;

        System.out.print("In " + days);
        System.out.print(" days light will travel about ");
        System.out.println(distance + " miles.");
    }
}
```

1.6 Compute the area of a circle.

```
public class Area {
```

```
    public static void main(String[] args) {
        double pi, r, a;

        r = 10.8;
        pi = 3.1416;
        a = pi * r * r;

        System.out.println("Area of circle is " + a);
    }
}
```

1.7 Demonstrate char data type.

```
public class CharDemo {

    public static void main(String[] args) {
        char ch1, ch2;

        ch1 = 88;
        ch2 = 'Y';

        System.out.print("ch1 and ch2: ");
        System.out.println(ch1 + " " + ch2);
    }
}
```

1.8 Char variables behave like integers.

```
public class CharDemo2 {

    public static void main(String[] args) {
        char ch1;

        ch1 = 'X';
        System.out.println("ch1 contains " + ch1);

        ch1++;
        System.out.println("ch1 is now " + ch1);
    }
}
```

1.9 Demonstrate boolean values.

```
public class BoolTest {

    public static void main(String[] args) {
        boolean b;

        b = false;
        System.out.println("b is " + b);
        b = true;
        System.out.println("b is " + b);

        if (b)
            System.out.println("This is executed.");

        b = false;
        if (b)
            System.out.println("This is not executed.");

        System.out.println("10 > 9 is " + (10 > 9));
    }
}
```

1.10 Demonstrate dynamic initialization

```
public class DynInit {

    public static void main(String[] args) {
        double a = 3.0, b = 4.0;

        double c = Math.sqrt(a * a + b * b);

        System.out.println("Hypotenuse is " + c);
    }
}
```

1.11 Demonstrate block scope

```
public class Scope {

    public static void main(String[] args) {
        int x;

        x = 10;
        if (x == 10) {
            int y = 20;

            System.out.println("x and y: " + x + " " + y);
            x = y * 2;
        }

        // y = 100; //Error! y not known here

        System.out.println("x is " + x);
    }
}
```

1.12 LifeTime

```
public class LifeTime {

    public static void main(String[] args) {
        int x;

        for (x = 0; x < 3; x++) {
            int y = -1;
            System.out.println("y is: " + y);
            y = 100;
            System.out.println("y is now: " + y);
        }
    }
}
```

1.13 Demonstrate casts

```
public class Conversion {

    public static void main(String[] args) {
        byte b;
        int i = 257;
        double d = 323.142;

        System.out.println("\nConversion of int to byte.");
        b = (byte) i;
        System.out.println("i and b " + i + " " + b);

        System.out.println("\nConversion of double to int.");
        i = (int) d;
        System.out.println("d and i " + d + " " + i);

        System.out.println("\nConversion of double to byte.");
        b = (byte) d;
        System.out.println("d and b " + d + " " + b);
    }
}
```

1.14 Promote

```
public class Promote {

    public static void main(String[] args) {
        byte b = 42;
        char c = 'a';
        short s = 1024;
        int i = 50000;
        float f = 5.67f;
        double d = .1234;
        double result = (f * b) + (i / c) - (d * s);
        System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
        System.out.println("result = " + result);
    }
}
```

1.15 Demonstrate a one-dimensional array.

```
public class Array {

    public static void main(String[] args) {
        int month_days[];
        month_days = new int[12];
        month_days[0] = 31;
        month_days[1] = 28;
        month_days[2] = 31;
        month_days[3] = 30;
        month_days[4] = 31;
        month_days[5] = 30;
        month_days[6] = 31;
        month_days[7] = 31;
        month_days[8] = 30;
        month_days[9] = 31;
        month_days[10] = 30;
        month_days[10] = 31;
        System.out.println("April has " + month_days[3] + " days.");
    }
}
```

1.16 An improved version of the one-dimensional array

```
public class AutoArray {

    public static void main(String[] args) {
        int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
            31 };
        System.out.println("April has " + month_days[12] + " days.");
    }
}
```

1.17 Average an array of values.

```
public class Average {

    public static void main(String[] args) {
        double nums[] = { 10.1, 11.2, 12.3, 13.4, 14.5 };
        double result = 0;
        int i;
        for (i = 0; i < 5; i++) {
            result = result + nums[i];
        }
        System.out.println("Average is " + result / 5);
    }
}
```

1.18 Demonstrate a two-dimensional array.

```
public class TwoDArray {

    public static void main(String[] args) {
        int twoD[][] = new int[4][5];
        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 5; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

1.19 Manually allocate differing size second dimensions.

```
public class TwoDAgain {

    public static void main(String[] args) {
        int twoD[][] = new int[4][];
        twoD[0] = new int[1];
        twoD[1] = new int[2];
        twoD[2] = new int[3];
        twoD[3] = new int[4];

        int i, j, k = 0;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                twoD[i][j] = k;
                k++;
            }
        }

        for (i = 0; i < 4; i++) {
            for (j = 0; j < i + 1; j++) {
                System.out.print(twoD[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

1.20 Initialize a two-dimensional array.

```
public class Matrix {

    public static void main(String[] args) {
        double m[][] = { { 0 * 0, 1 * 0, 2 * 0, 3 * 0 }, { 0 * 1, 1 * 1,
            2 * 1, 3 * 1 }, { 0 * 2, 1 * 2, 2 * 2, 3 * 2 },
                { 0 * 3, 1 * 3, 2 * 3, 3 * 3 } };
        int i, j;

        for (i = 0; i < 4; i++) {
            for (j = 0; j < 4; j++) {
                System.out.print(m[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

1.21 Demonstrate a three-dimensional array.

```
public class ThreeDMatrix {

    public static void main(String[] args) {
        int threeD[][][] = new int[3][4][5];
        int i, j, k;

        for (i = 0; i < 3; i++) {
            for (j = 0; j < 4; j++) {
                for (k = 0; k < 5; k++) {
                    threeD[i][j][k] = i * j * k;
                }
            }
        }

        for (i = 0; i < 3; i++) {
            for (j = 0; j < 4; j++) {
                for (k = 0; k < 5; k++) {
                    System.out.print(threeD[i][j][k] + " ");
                }
                System.out.println();
            }
            System.out.println();
        }
    }
}
```

1.22 A simple demonstration of local variable type inference.

```
public class VarDemo {

    public static void main(String[] args) {
        var avg = 10.0;
        System.out.println("Value of avg: " + avg);

        int var = 1;
        System.out.println("Value of var: " + var);

        var k = -var;
        System.out.println("Value of k: " + k);
    }
}
Footer
```

1.23 Demonstrate the basic arithmetic operators

```
public class BasicMath {

    public static void main(String[] args) {
        System.out.println("Integer Arithmetic");
        int a = 1 + 1;
        int b = a * 3;
        int c = b / 4;
        int d = c - a;
        int e = -d;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
        System.out.println("e = " + e);

        System.out.println("\nFloating Point Arithmetic");
        double da = 1 + 1;
        double db = da * 3;
        double dc = db / 4;
        double dd = dc - da;
        double de = -dd;
        System.out.println("da = " + da);
        System.out.println("db = " + db);
        System.out.println("dc = " + dc);
        System.out.println("dd = " + dd);
        System.out.println("de = " + de);
    }
```

```
}
```

1.24 Demonstrate the % operator.

```
public class Modulus {

    public static void main(String[] args) {
        int x = 42;
        double y = 42.25;

        System.out.println("x mod 10 = " + x % 10);
        System.out.println("y mod 10 = " + y % 10);
    }
```

1.25 Demonstrate several assignment operators

```
public class OpEquals {

    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;

        a += 5;
        b *= 4;
        c += a * b;
        c %= 6;

        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}
```

1.26 Demonstrate ++.

```
public class IncDec {

    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c;
        int d;

        c = ++b;
        d = a++;
        c++;

        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
        System.out.println("d = " + d);
    }
}
```

1.27 Demonstrate the bitwise logical operators

```
public class BitLogic {

    public static void main(String[] args) {
        String binary[] = { "0000", "0001", "0010", "0011", "0100",
            "0101", "0110", "0111", "1000", "1001", "1010",
               "1011", "1100", "1101", "1110", "1111" };
        int a = 3;
        int b = 6;
        int c = a | b;
        int d = a & b;
        int e = a ^ b;
        int f = (~a & b) | (a & ~b);
        int g = ~a & 0x0f;

        System.out.println("     a = " + binary[a]);
        System.out.println("     b = " + binary[b]);
        System.out.println("   a|b = " + binary[c]);
        System.out.println("   a&b = " + binary[d]);
        System.out.println("   a^b = " + binary[e]);
        System.out.println("~a&b|a&~b = " + binary[f]);
        System.out.println("    ~a = " + binary[g]);
    }
}
```

1.28 Left shifting a byte value.

```
public class ByteShift {

    public static void main(String[] args) {
        byte a = 64, b;
        int i;

        i = a << 2;
        b = (byte) (a << 2);

        System.out.println("Original value of a: " + a);
        System.out.println("i and b: " + i + " " + b);
    }
}
```

1.29 Left shifting as a quick way to multiply by 2

```
public class MultByTwo {

    public static void main(String[] args) {
        int i;
        int num = 0xFFFFFFE;

        for (i = 0; i < 4; i++) {
            num = num << 1;
            System.out.println(num);
        }
    }
}
```

1.30 Masking sign extension

```
public class HexByte {

    public static void main(String[] args) {
        char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
            'a', 'b', 'c', 'd', 'e', 'f' };

        byte b = (byte) 0xf1;
        System.out.println("b = 0x" + hex[(b >> 4) & 0x0f] + hex[b &
            0x0f]);
    }
}
```

1.31 Unsigned shifting a byte value

```
public class ByteUShift {

    public static void main(String[] args) {
        char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
            'a', 'b', 'c', 'd', 'e', 'f' };
        byte b = (byte) 0xf1;
        byte c = (byte) (b >> 4);
        byte d = (byte) (b >>> 4);
        byte e = (byte) ((b & 0xff) >> 4);

        System.out.println("            b = 0x" + hex[(b >> 4) & 0x0f] +
            hex[b & 0x0f]);
        System.out.println("       b >> 4 = 0x" + hex[(c >> 4) & 0x0f] +
            hex[c & 0x0f]);
        System.out.println("      b >>> 4 = 0x" + hex[(d >> 4) & 0x0f] +
            hex[d & 0x0f]);
        System.out.println(" (b & 0xff) >> 4 = 0x" + hex[(e >> 4) &
            0x0f] + hex[e & 0x0f]);
    }
}
```

1.32 OpBitEquals

```
public class OpBitEquals {

    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;

        a |= 4;
        b >>= 1;
        c <<= 1;
        a ^= c;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}
```

1.33 Demonstrate the boolean logical operators.

```
public class BoolLogic {

    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;
        boolean c = a | b;
        boolean d = a & b;
        boolean e = a ^ b;
        boolean f = (!a & b) | (a & !b);
        boolean g = !a;
        System.out.println("     a = " + a);
        System.out.println("     b = " + b);
        System.out.println("   a|b = " + c);
        System.out.println("   a&b = " + d);
        System.out.println("   a^b = " + e);
        System.out.println("!a&b|a&!b = " + f);
        System.out.println("    !a = " + g);
    }
}
```

1.34 Demonstrate ?.

```
public class Ternary {

    public static void main(String[] args) {
        int i, k;

        i = 10;
        k = i < 0 ? -i : i;
        System.out.println("Absolute value of ");
        System.out.println(i + " is " + k);

        i = -10;
        k = i < 0 ? -i : i;
        System.out.println("Absolute value of ");
        System.out.println(i + " is " + k);
    }
}
```

1.35 Demonstrate if-else-if statements.

```
public class IfElse {

    public static void main(String[] args) {
        int month = 4;
        String season;

        if (month == 12 || month == 1 || month == 2) {
            season = "Winter";
        } else if (month == 3 || month == 4 || month == 5) {
            season = "Spring";
        } else if (month == 6 || month == 7 || month == 8) {
            season = "Summer";
        } else if (month == 9 || month == 10 || month == 11) {
            season = "Autumn";
        } else {
            season = "Bogus Month";
        }

        System.out.println("April is in the " + season + ".");
    }
}
```

1.36 A simple example of the switch.

```
public class SampleSwitch {

    public static void main(String[] args) {
        for (int i = 0; i < 6; i++) {
            switch (i) {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");
                break;
            case 2:
                System.out.println("i is two.");
                break;
            case 3:
                System.out.println("i is three.");
                break;

            default:
                System.out.println("i is greater than 3.");
            }
        }
    }
}
```

1.37 In a switch, break statements are optional.

```
public class MissingBreak {

    public static void main(String[] args) {
        for (int i = 0; i < 12; i++) {
            switch (i) {
            case 0:
            case 1:
            case 2:
            case 3:
            case 4:
                System.out.println("i is less than 5");
                break;
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
                System.out.println("i is less than 10");
                break;
            default:
                System.out.println("i is 10 or more");
```

```
            }
        }
    }
}
```

1.38 An improved version of the season program.

```
public class Switch {

    public static void main(String[] args) {
        int month = 4;

        String season;

        switch (month) {
        case 12:
        case 1:
        case 2:
            season = "Winter";
            break;
        case 3:
        case 4:
        case 5:
            season = "Spring";
            break;
        case 6:
        case 7:
        case 8:
            season = "Summer";
            break;
        case 9:
        case 10:
        case 11:
            season = "Autumn";
            break;
        default:
            season = "Bogus Month";
        }

        System.out.println("April is in the " + season + ".");
    }
}
```

1.39 Use a string to control a switch statement.

```
public class StringSwitch {
```

```
    public static void main(String[] args) {
        String str = "two";

        switch (str) {
        case "one":
            System.out.println("one");
            break;
        case "two":
            System.out.println("two");
            break;
        case "three":
            System.out.println("three");
            break;
        default:
            System.out.println("no match");
            break;
        }
    }
}
```

1.40 Demonstrate the while loop.

```
public class While {

    public static void main(String[] args) {
        int n = 10;

        while (n > 0) {
            System.out.println("tick " + n);
            n--;
        }
    }
}
```

1.41 The target of a loop can be empty.

```
public class NoBody {

    public static void main(String[] args) {
        int i, j;

        i = 100;
        j = 200;

        while (++i < --j)
            ;

        System.out.println("Midpoint is " + i);
    }
}
```

1.42 Demonstrate the do-while loop.

```
public class DoWhile {

    public static void main(String[] args) {
        int n = 10;

        do {
            System.out.println("tick " + n);
        } while (--n > 0);
    }
}
```

1.43 Using a do-while to process a menu selection.

```
public class Menu {

    public static void main(String[] args) throws IOException {
        char choice;

        do {
            System.out.println("Help on: ");
            System.out.println(" 1. if");
            System.out.println(" 2. switch");
            System.out.println(" 3. while");
            System.out.println(" 4. do-while");
            System.out.println(" 5. for\n");
            System.out.println("Choose one:");
            choice = (char) System.in.read();
        } while (choice < '1' || choice > '5');

        System.out.println("\n");

        switch (choice) {
        case '1':
            System.out.println("The if:\n");
            System.out.println("if(condition) statement;");
            System.out.println("else statement;");
            break;
        case '2':
            System.out.println("The switch:\n");
            System.out.println("switch(express) {");
            System.out.println("case constant:");
            System.out.println("  statement sequence");
            System.out.println("  break;");
            System.out.println("  //...");
            System.out.println("}");
            break;
        case '3':
            System.out.println("The while:\n");
            System.out.println("while(condition) statement;");
            break;
        case '4':
            System.out.println("The do-while:\n");
            System.out.println("do {");
            System.out.println("  statement;");
            System.out.println("} while (condition);");
            break;
        case '5':
            System.out.println("The for:\n");
            System.out.println("for(init; condition; iteration)");
            System.out.println("  statement;");
```

```
            break;
        }
    }
}
```

1.44 Demonstrate the for loop.

```
public class ForTick {

    public static void main(String[] args) {
        for (int n = 10; n > 0; n--) {
            System.out.println("tick " + n);
        }
    }
}
```

1.45 Test for primes

```
public class FindPrime {

    public static void main(String[] args) {
        int num;
        boolean isPrime;

        num = 14;

        if (num < 2)
            isPrime = false;
        else
            isPrime = true;

        for (int i = 2; i <= num / i; i++) {
            if ((num % i) == 0) {
                isPrime = false;
                break;
            }
        }

        if (isPrime)
            System.out.println("Prime");
        else
            System.out.println("Not Prime");
    }
}
```

1.46 Sample For Loop Example

```
public class Sample {

    public static void main(String[] args) {
        int a, b;

        b = 4;
        for (a = 1; a < b; a++) {
            System.out.println("a = " + a);
            System.out.println("b = " + b);
            b--;
        }
    }
}
```

1.47 Using the comma

```
public class Comma {

    public static void main(String[] args) {
        int a, b;

        for (a = 1, b = 4; a < b; a++, b--) {
            System.out.println("a = " + a);
            System.out.println("b = " + b);
        }
    }
}
```

1.48 Parts of the for loop can be empty.

```
public class ForVar {

    public static void main(String[] args) {
        int i;
        boolean done = false;

        i = 0;
        for (; !done;) {
            System.out.println("i is " + i);
            if (i == 10)
                done = true;
            i++;
        }
    }
}
```

1.49 Use a for-each style for loop.

```
public class ForEach {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum = 0;

        for (int x : nums) {
            System.out.println("Value is: " + x);
            sum += x;
        }

        System.out.println("Summation: " + sum);
    }
}
```

1.50 ForEach2

```
public class ForEach2 {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int sum = 0;

        for (int x : nums) {
            System.out.println("Value is: " + x);
            sum += x;
            if (x == 5)
                break;
        }

        System.out.println("Summation of first 5 elements: " + sum);
    }
}
```

1.51 The for-each loop is essentially read-only.

```
public class NoChange {

    public static void main(String[] args) {
        int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        for (int x : nums) {
            System.out.print(x + " ");
            x = x * 10;
        }

        System.out.println();

        for (int x : nums) {
            System.out.print(x + " ");
        }

        System.out.println();
    }
}
```

1.52 Use for-each style for on a two-dimensional array.

```
public class ForEach3 {
```

```
    public static void main(String[] args) {
        int sum = 0;
        int nums[][] = new int[3][5];

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 5; j++) {
                nums[i][j] = (i + 1) * (j + 1);
            }
        }

        for (int x[] : nums) {
            for (int y : x) {
                System.out.println("Value is: " + y);
                sum += y;
            }
        }

        System.out.println("Summation: " + sum);
    }
}
Footer
```

1.53 Search an array using for-each style for

```
public class Search {

    public static void main(String[] args) {
        int nums[] = { 6, 8, 3, 7, 5, 6, 1, 4 };
        int val = 5;
        boolean found = false;

        for (int x : nums) {
            if (x == val) {
                found = true;
                break;
            }
        }

        if (found) {
            System.out.println("Value found!");
        }
    }
}
```

1.54 Use type inference in a for loop.

```
public class TypeInferenceInFor {

    public static void main(String[] args) {
        System.out.println("Values of x: ");
        for (var x = 2.5; x < 100.0; x = x * 2)
            System.out.print(x + " ");

        System.out.println();

        int[] nums = { 1, 2, 3, 4, 5, 6 };
        System.out.println("Values in nums array: ");
        for (var v : nums)
            System.out.print(v + " ");

        System.out.println();
    }
}
```

1.55 Loops may be nested.

```
public class Nested {

    public static void main(String[] args) {
        int i, j;

        for (i = 0; i < 10; i++) {
            for (j = i; j < 10; j++) {
                System.out.print(".");
            }
            System.out.println();
        }
    }
}
```

1.56 Using break to exit a loop.

```
public class BreakLoop {

    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            if (i == 10) {
                break;
            }
            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
Footer
```

1.57 Using break to exit a while loop.

```
public class BreakLoop2 {

    public static void main(String[] args) {
        int i = 0;

        while (i < 100) {
            if (i == 10) {
                break;
            }
            System.out.println("i: " + i);
            i++;
        }
        System.out.println("Loop complete.");
    }
}
Footer
```

1.58 Using break with nested loops.

```
public class BreakLoop3 {

    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; j < 100; j++) {
                if (j == 10) {
                    break;
                }
                System.out.print(j + " ");
            }
            System.out.println();
        }
        System.out.println("Loops complete.");
    }
}
Footer
```

1.59 Breaking Blocks

```
public class Break {

    public static void main(String[] args) {
        boolean t = true;

        first: {
            second: {
                third: {
                    System.out.println("Before the break.");
                    if (t)
                        break second;
                    System.out.println("this won't execute");
                }
                System.out.println("this won't execute");
            }
            System.out.println("This is after second block.");
        }
    }
}
```

1.60 Using break to exit from nested loops.

```
public class BreakLoop4 {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
            for (int j = 0; i < 100; j++) {
                if (j == 10)
                    break outer;
                System.out.print(j + " ");
            }
            System.out.println("This will not print");
        }
        System.out.println("Loops complete.");
    }
}
```

1.61 This program contains an error.

```
public class BreakErr {

    public static void main(String[] args) {
        one: for (int i = 0; i < 3; i++) {
            System.out.print("Pass " + i + ": ");
        }

        for (int j = 0; j < 100; j++) {
            if (j == 10)
                break one; // WRONG
            System.out.print(j + " ");
        }
    }
}
```

1.62 Demonstrate continue.

```
public class Continue {

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
            if (i % 2 == 0)
                continue;
            System.out.println("");
        }
    }
}
```

1.63 Using continue with a label.

```
public class ContinueLabel {

    public static void main(String[] args) {
        outer: for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 10; j++) {
                if (j > i) {
                    System.out.println();
                    continue outer;
                }
                System.out.print(" " + (i * j));
            }
            System.out.println();
        }
    }
}
```

1.64 Demonstrate return.

```
public class Return {

    public static void main(String[] args) {
        boolean t = true;

        System.out.println("Before the return.");

        if (t) {
            return;
        }

        System.out.println("This won't execute");
    }
}
```

2 Classes, objects, Arrays and Strings

- Classes Objects: Reference Variables, Passing parameters to Methods and Returning parameters from the methods, Static members, Non-Static members Nested and Inner Classes. Static Initialization Block(SIB), Instance Initialization Block(IIB)
- Constructors: Parameterized Constructors, chaining of constructor, finalize() Method, Method overloading, Constructors Overloading.
- Recursion,
- Command-Line Arguments
- Wrapper classes,
- InputBufferReader, OutputBufferReader,
- String Buffer classes, String functions.
- Arrays Vectors: One and Two Dimensional arrays, Irregular arrays, dynamic arrays, Array List and Array of Object.

2.1 BoxClass

```
class Box {
    double width;
    double height;
    double depth;

    Box() {
        System.out.println("Constructing Box");
        width = 10;
        height = 10;
        depth = 10;
    }

    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    void volume() {
        System.out.print("Volume is ");
        System.out.println(width * height * depth);
    }

    double computeVolume() {
        return width * height * depth;
    }

    void setDim(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
}
```

```
// This class declares an object of type Box.
public class BoxDemo {
    public static void main(String[] args) {
        Box mybox = new Box();
        double vol;

        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;

        vol = mybox.width * mybox.height * mybox.depth;
```

```
        System.out.println("Volume is " + vol);
    }
}
```

```
// This program declares two Box objects.
public class BoxDemo2 {

    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;

        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;

        vol = mybox1.width * mybox1.height * mybox1.depth;
        System.out.println("Volume is " + vol);

        vol = mybox2.width * mybox2.height * mybox2.depth;
        System.out.println("Volume is " + vol);

    }
}
```

```
//This program includes a method inside the box class.
public class BoxDemo3 {

    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();

        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;

        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;

        mybox1.volume();

        mybox2.volume();
    }
```

```
}
```

```
// Compute Volume return values
public class BoxDemo4 {

    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;

        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;

        vol = mybox1.computeVolume();
        System.out.println("Volume is " + vol);

        vol = mybox2.computeVolume();
        System.out.println("Volume is " + vol);
    }
}
```

2.2 This program uses a parameterized method.

```
public class BoxDemo5 {

    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        mybox1.setDim(10, 20, 15);
        mybox2.setDim(3, 6, 9);

        vol = mybox1.computeVolume();
        System.out.println("Volume is " + vol);

        vol = mybox2.computeVolume();
        System.out.println("Volume is " + vol);
    }
}
```

```
// Box uses a constructor to initialize the dimensions of a box.
public class BoxDemo6 {

    public static void main(String[] args) {
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;

        vol = mybox1.computeVolume();
        System.out.println("Volume is " + vol);

        vol = mybox2.computeVolume();
        System.out.println("Volume is " + vol);
    }
}
```

```
// Box uses a parameterized constructor to initialize the dimensions of
    a box.
public class BoxDemo7 {

    public static void main(String[] args) {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box(3, 6, 9);
        double vol;

        vol = mybox1.computeVolume();
        System.out.println("Volume is " + vol);

        vol = mybox2.computeVolume();
        System.out.println("Volume is " + vol);
    }
}
```

2.3 Stock Demo

```
class Stack {
    int stck[] = new int[10];
    int tos;

    Stack() {
        tos = -1;
    }
```

```
    void push(int item) {
        if (tos == 9) {
            System.out.println("Stack is full.");
        } else {
            stck[++tos] = item;
        }
    }

    int pop() {
        if (tos < 0) {
            System.out.println("Stack underflow.");

            return 0;
        } else {
            return stck[tos--];
        }
    }
}
```

```
public class TestStack {

    public static void main(String[] args) {
        Stack mystack1 = new Stack();
        Stack mystack2 = new Stack();

        for (int i = 0; i < 10; i++)
            mystack1.push(i);
        for (int i = 10; i < 20; i++)
            mystack2.push(i);

        System.out.println("Stack in mystack1:");
        for (int i = 0; i < 10; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stack in mystack2:");
        for (int i = 0; i < 10; i++)
            System.out.println(mystack2.pop());
    }
}
```

2.4 Demonstrate method overloading

```
class OverloadDemo {
    void test() {
        System.out.println("No parameters");
    }
```

```
    void test(int a) {
        System.out.println("a: " + a);
    }

    void test(int a, int b) {
        System.out.println("a and b: " + a + " " + b);
    }

    double test(double a) {
        System.out.println("double a: " + a);
        return a * a;
    }
}


public class Overload {
    public static void main(String[] args) {
        OverloadDemo ob = new OverloadDemo();
        double result;

        ob.test();
        ob.test(10);
        ob.test(10, 20);
        result = ob.test(123.25);
        System.out.println("Result of ob.test(123.25): " + result);
    }
}
```

2.5 Object may be passed to methods.

```
class Test {
    int a, b;

    Test(int i, int j) {
        a = i;
        b = j;
    }

    boolean equalTo(Test o) {
        if (o.a == a && o.b == b) {
            return true;
        }
        return false;
    }
}

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

```
        Test ob1 = new Test(100, 22);
        Test ob2 = new Test(100, 22);
        Test ob3 = new Test(-1, -1);

        System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
        System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
    }
}
```

2.6 Constructor Overloading 1

```
class Box {
    double width;
    double height;
    double depth;

    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    Box() {
        width = -1;
        height = -1;
        depth = -1;
    }

    Box(double len) {
        width = height = depth = len;
    }

    double volume() {
        return width * height * depth;
    }
}

public class OverloadCons {
    public static void main(String[] args) {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);

        double vol;

        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
```

```
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);

        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
    }

}
```

2.7 Constructor Overloading 2 - Box allows one object to initialize another

```
class Box2 {
    double width;
    double height;
    double depth;

    Box2(Box2 ob) {
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }

    Box2(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    Box2() {
        width = -1;
        height = -1;
        depth = -1;
    }

    Box2(double len) {
        width = height = depth = len;
    }

    double volume() {
        return width * height * depth;
    }
}

public class OverloadCons2 {
    public static void main(String[] args) {
        Box2 mybox1 = new Box2(10, 20, 15);
        Box2 mybox2 = new Box2();
        Box2 mycube = new Box2(7);

        Box2 myclone = new Box2(mybox1);

        double vol;

        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);

        vol = mybox2.volume();
```

```
        System.out.println("Volume of mybox2 is " + vol);

        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);

        vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);
    }

}
```

2.8 Primitive types are passed by value.

```
class Test2 {
    void meth(int i, int j) {
        i *= 2;
        j /= 2;
    }
}

class CallByValue {

    public static void main(String[] args) {
        Test2 ob = new Test2();

        int a = 15, b = 20;

        System.out.println("a and b before call: " + a + " " + b);

        ob.meth(a, b);

        System.out.println("a and b after call: " + a + " " + b);
    }
}
```

2.9 Objects are passed through their references.

```
class Test3 {
    int a, b;

    Test3(int i, int j) {
        a = i;
        b = j;
    }

    void meth(Test3 o) {
        o.a *= 2;
        o.b /= 2;
    }
}

public class PassObjRef {

    public static void main(String[] args) {
        Test3 ob = new Test3(15, 20);

        System.out.println("a and b before call: " + ob.a + " " + ob.b);

        ob.meth(ob);

        System.out.println("a and b after call: " + ob.a + " " + ob.b);
    }
}
```

2.10 Returning an object.

```
class Test4 {
    int a;

    Test4(int i) {
        a = i;
    }

    Test4 incrByTen() {
        Test4 temp = new Test4(a + 10);
        return temp;
    }
}

public class RetOb {
    public static void main(String[] args) {
        Test4 ob1 = new Test4(2);
```

```
        Test4 ob2;

        ob2 = ob1.incrByTen();
        System.out.println("ob1.a: " + ob1.a);
        System.out.println("ob2.a: " + ob2.a);

        ob2 = ob2.incrByTen();
        System.out.println("ob2.a after second increase: " + ob2.a);
    }
}
```

2.11 A simple example of recursion.

```
class Factorial {

    int fact(int n) {
        int result;

        if (n == 1) {
            return 1;
        }

        result = fact(n - 1) * n;

        return result;
    }
}

class Recursion {

    public static void main(String[] args) {
        Factorial f = new Factorial();

        System.out.println("Factorial of 3 is " + f.fact(3));
        System.out.println("Factorial of 4 is " + f.fact(4));
        System.out.println("Factorial of 5 is " + f.fact(5));
    }
}
```

2.12 Another example that uses recursion.

```
class RecTest {
    int values[];

    RecTest(int i) {
        values = new int[i];
    }

    void printArray(int i) {
        if (i == 0) {
            return;
        } else {
            printArray(i - 1);
        }
        System.out.println("[" + (i - 1) + "] " + values[i - 1]);
    }
}

class Recursion2 {
    public static void main(String[] args) {
        RecTest ob = new RecTest(10);
        int i;

        for (i = 0; i < 10; i++)
            ob.values[i] = i;

        ob.printArray(10);
    }
}
```

2.13 This program demonstrates the difference between public and private.

```
class Test5 {
    int a;
    public int b;
    private int c;

    void setc(int i) {
        c = i;
    }

    int getc() {
        return c;
    }
}

public class AccessTest {

    public static void main(String[] args) {
        Test5 ob = new Test5();

        ob.a = 10;
        ob.b = 20;

        // ob.c = 100;

        ob.setc(100);
        System.out.println("a, b, and c: " + ob.a + " " + ob.b + " " +
            ob.getc());
    }
}
```

2.14 This class defines an integer stack that can hold 10 values.

```
class Stack {
    private int stck[] = new int[10];
    private int tos;

    Stack() {
        tos = -1;
    }

    void push(int item) {
        if (tos == 9) {
            System.out.println("Stack is full.");
```

```
        } else {
            stck[++tos] = item;
        }
    }

    int pop() {
        if (tos < 0) {
            System.out.println("Stack underflow.");

            return 0;
        } else {
            return stck[tos--];
        }
    }
}

class TestStack {

    public static void main(String[] args) {
        Stack mystack1 = new Stack();
        Stack mystack2 = new Stack();

        for (int i = 0; i < 10; i++)
            mystack1.push(i);
        for (int i = 10; i < 20; i++)
            mystack2.push(i);

        System.out.println("Stack in mystack1:");
        for (int i = 0; i < 10; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stack in mystack2:");
        for (int i = 0; i < 10; i++)
            System.out.println(mystack2.pop());

        // mystack1.tos = -2;
        // mystack1.stck[3] = 200;
    }
}
```

2.15 Demonstrate static variables, methods, and blocks.

```
public class UseStatic {
    static int a = 3;
    static int b;

    static void meth(int x) {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }

    static {
        System.out.println("Static block initialized.");
        b = a * 4;
    }

    public static void main(String args[]) {
        meth(42);
    }
}
```

2.16 Static Variables Demo

```
class StaticDemo {
    static int a = 42;
    static int b = 99;

    static void callme() {
        System.out.println("a = " + a);
    }
}

public class StaticByName {

    public static void main(String[] args) {
        StaticDemo.callme();
        System.out.println("b = " + StaticDemo.b);
    }
}
```

2.17 This program demonstrates the length array member.

```
public class Length {

    public static void main(String[] args) {
        int a1[] = new int[10];
        int a2[] = { 3, 5, 7, 1, 8, 99, 44, -10 };
        int a3[] = { 4, 3, 2, 1 };

        System.out.println("length of a1 is " + a1.length);
        System.out.println("length of a2 is " + a2.length);
        System.out.println("length of a3 is " + a3.length);
    }
}
```

2.18 Improved Stack class that uses the length array member.

```
class Stack2 {
    private int stck[];
    private int tos;

    Stack2(int size) {
        stck = new int[size];
        tos = -1;
    }

    void push(int item) {
        if (tos == stck.length - 1) {
            System.out.println("Stack is full.");
        } else {
            stck[++tos] = item;
        }
    }

    int pop() {
        if (tos < 0) {
            System.out.println("Stack underflow.");

            return 0;
        } else {
            return stck[tos--];
        }
    }
}
```

```
public class TestStack2 {

    public static void main(String[] args) {
        Stack2 mystack1 = new Stack2(5);
        Stack2 mystack2 = new Stack2(8);

        for (int i = 0; i < 5; i++)
            mystack1.push(i);
        for (int i = 0; i < 8; i++)
            mystack2.push(i);

        System.out.println("Stack in mystack1:");
        for (int i = 0; i < 5; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stack in mystack2:");
        for (int i = 0; i < 8; i++)
            System.out.println(mystack2.pop());
    }
}
```

2.19 Inner and Outer Class Demo

```
class Outer {
    int outer_x = 100;

    void test() {
        Inner inner = new Inner();
        inner.display();
    }

    class Inner {
        int y = 10;

        void display() {
            System.out.println("display: outer_x = " + outer_x);
        }
    }

    void showy() {
        // System.out.println(y);
    }
}

public class InnerClassDemo {

    public static void main(String[] args) {
        Outer outer = new Outer();
```

```
        outer.test();
    }
}
```

2.20 Define an inner class within a for loop.

```
class Outer2 {
    int outer_x = 100;

    void test() {
        for (int i = 0; i < 10; i++) {
            class Inner2 {
                void display() {
                    System.out.println("display: outer_x = " + outer_x);
                }
            }
            Inner2 inner = new Inner2();
            inner.display();
        }
    }
}

public class InnerClassDemo2 {

    public static void main(String[] args) {
        Outer2 outer = new Outer2();
        outer.test();
    }
}
```

2.21 Demonstrating strings.

```
public class StringDemo {

    public static void main(String[] args) {
        String strOb1 = "First String";
        String strOb2 = "Second String";
        String strOb3 = strOb1 + " and " + strOb2;

        System.out.println(strOb1);
        System.out.println(strOb2);
        System.out.println(strOb3);
    }
}
```

2.22 Demonstrating some String methods.

```
public class StringDemo2 {

    public static void main(String[] args) {
        String strOb1 = "First String";
        String strOb2 = "Second String";
        String strOb3 = strOb1;

        System.out.println("Length of strOb1: " + strOb1.length());
        System.out.println("Char at index 3 in strOb1: " +
            strOb1.charAt(3));

        if (strOb1.equals(strOb2)) {
            System.out.println("strOb1 == strOb2");
        } else {
            System.out.println("strOb1 != strOb2");
        }

        if (strOb1.equals(strOb3)) {
            System.out.println("strOb1 == strOb3");
        } else {
            System.out.println("strOb1 != strOb3");
        }
    }
}
```

2.23 Demonstrate String arrays

```
public class StringDemo3 {

    public static void main(String[] args) {
        String str[] = { "one", "two", "three" };

        for (int i = 0; i < str.length; i++) {
            System.out.println("str[" + i + "]: " + str[i]);
        }
    }
}
```

2.24 Display all command-line arguments.

```
public class CommandLine {

    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println("args[" + i + "]: " + args[i]);
    }
}
```

2.25 Demonstrate variable-length arguments.

```
public class VarArgs {
    static void vaTest(int... v) {
        System.out.print("Number of args: " + v.length + " Contents: ");
        for (int x : v) {
            System.out.print(x + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n1[] = { 10 };
        int n2[] = { 1, 2, 3 };
        int n3[] = { 10 };

        vaTest(n1);
        vaTest(n2);
        vaTest(n3);
    }
}
```

2.26 Use varargs with standard arguments.

```
public class VarArgs2 {

    static void vaTest(String msg, int... v) {
        System.out.print(msg + v.length + " Contents: ");

        for (int x : v)
            System.out.print(x + " ");

        System.out.println();
    }
```

```
    public static void main(String[] args) {
        vaTest("One vararg: ", 10);
        vaTest("Three varargs: ", 1, 2, 3);
        vaTest("No varargs: ");

    }
}
```

2.27 Varargs and overloading.

```
public class VarArgs3 {

    static void vaTest(int... v) {
        System.out.print("vaTest(int ...): " + "Number of args: " +
            v.length + " Contents: ");

        for (int x : v) {
            System.out.print(x + " ");
        }

        System.out.println();
    }

    static void vaTest(boolean... v) {
        System.out.print("vaTest(boolean ...): " + "Number of args: " +
            v.length + " Contents: ");

        for (boolean x : v) {
            System.out.print(x + " ");
        }

        System.out.println();
    }

    static void vaTest(String msg, int... v) {
        System.out.print("vaTest(String, int ...): " + msg + v.length +
            " Contents: ");

        for (int x : v) {
            System.out.print(x + " ");
        }

        System.out.println();
    }

    public static void main(String[] args) {
        vaTest(1, 2, 3);
        vaTest("Testing: ", 10, 20);
```

```
        vaTest(true, false, false);
    }
}
```

2.28 Varargs, overloading, and ambiguity.

```
public class VarArgs4 {

    static void vaTest(int... v) {
        System.out.print("vaTest(int ...): " + "Number of args: " +
             v.length + " Contents: ");

        for (int x : v) {
            System.out.print(x + " ");
        }

        System.out.println();
    }

    static void vaTest(boolean... v) {
        System.out.print("vaTest(boolean ...): " + "Number of args: " +
             v.length + " Contents: ");

        for (boolean x : v) {
            System.out.print(x + " ");
        }

        System.out.println();
    }

    public static void main(String[] args) {
        vaTest(1, 2, 3);
        vaTest(true, false, false);
        // vaTest();
    }
}
```

2.29 Local variable type inference with a user-defined class type.

```
class MyClass {
    private int i;

    MyClass(int k) {
        i = k;
    }

    int geti() {
        return i;
    }

    void seti(int k) {
        if (k >= 0)
            i = k;
    }
}

public class RefVarDemo {

    public static void main(String[] args) {
        var mc = new MyClass(10);

        System.out.println("Value of i in mc is " + mc.geti());
        mc.seti(19);
        System.out.println("Value of i in mc is now " + mc.geti());
    }
}
```

3 Module 3 : Inheritance, Packages and Interfaces

- Inheritance: Inheritance Basics, Types of Inheritance in Java, member access, using Super- to call superclass Constructor, to access member of super class(variables and methods), creating multilevel hierarchy, Constructors in inheritance, method overriding, Abstract classes and methods, using final, Dynamic Method Dispatch
- Packages: Defining packages, creating packages and Importing and accessing packages
- Interfaces: Defining, implementing and extending interfaces, variables in interfaces, Default Method in Interface ,Static Method in interface, Abstract Classes vs Interfaces.

3.1 A simple example of inheritance.

```
class A {
    int i, j;

    void showij() {
        System.out.println("i and j: " + i + " " + j);
    }
}

class B extends A {
    int k;

    void showk() {
        System.out.println("k: " + k);
    }

    void sum() {
        System.out.println("i+j+k: " + (i + j + k));
    }
}

public class SimpleInheritance {

    public static void main(String[] args) {
        A superOb = new A();
        B subOb = new B();

        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();

        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb: ");
        subOb.showij();
        subOb.showk();
        System.out.println();

        System.out.println("Sum of i, j and k in subOb:");
        subOb.sum();
    }
}
```

3.2 In a class hierarchy, private members remain private to their class.

```
class A2 {
    int i;
    private int j;

    void setij(int x, int y) {
        i = x;
        j = y;
    }
}

class B2 extends A2 {
    int total;

    void sum() {
        // total = i + j;
    }
}

public class Access {

    public static void main(String[] args) {
        B2 subOb = new B2();

        subOb.setij(10, 12);

        subOb.sum();
        System.out.println("Total is " + subOb.total);
    }
}
```

3.3 This program uses inheritance to extend Box.

```
class Box {
    private double width;
    private double height;
    private double depth;

    Box(Box ob) {
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }

    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

    Box() {
        width = -1;
        height = -1;
        depth = -1;
    }

    Box(double len) {
        width = height = depth = len;
    }

    double volume() {
        return width * height * depth;
    }
}
```

```
class BoxWeight extends Box {
    double weight;

    BoxWeight(BoxWeight ob) {
        super(ob);
        weight = ob.weight;
    }

    BoxWeight(double w, double h, double d, double m) {
        super(w, h, d);
        weight = m;
    }

    BoxWeight() {
```

```
        super();
        weight = -1;
    }

    BoxWeight(double len, double m) {
        super(len);
        weight = m;
    }
}
```

```
public class DemoBoxWeight {

    public static void main(String[] args) {
        BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
        BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
        double vol;

        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        System.out.println("Weight of mybox1 is " + mybox1.weight);
        System.out.println();

        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        System.out.println("Weight of mybox2 is " + mybox2.weight);
        System.out.println();
    }
}
```

3.4 RefDemo

```
public class RefDemo {

    public static void main(String[] args) {
        BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
        Box plainbox = new Box();
        double vol;

        vol = weightbox.volume();
        System.out.println("Volume of weightbox is " + vol);
        System.out.println("Weight of weightbox is " + weightbox.weight);
        System.out.println();

        plainbox = weightbox;

        vol = plainbox.volume();
        System.out.println("Volume of plainbox is " + vol);

        // System.out.println("Weight of plainbox is" + plainbox.weight);
    }
}
```

3.5 Super Demo

This class uses the Box and BoxWeight classes defined earlier

```
public class DemoSuper {

    public static void main(String[] args) {
        BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
        BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
        BoxWeight mybox3 = new BoxWeight();
        BoxWeight mycube = new BoxWeight(3, 2);
        BoxWeight myclone = new BoxWeight(mybox1);
        double vol;

        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        System.out.println("Weight of mybox1 is " + mybox1.weight);
        System.out.println();

        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        System.out.println("Weight of mybox2 is " + mybox2.weight);
        System.out.println();

        vol = mybox3.volume();
        System.out.println("Volume of mybox3 is " + vol);
        System.out.println("Weight of mybox3 is " + mybox3.weight);
        System.out.println();

        vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);
        System.out.println("Weight of myclone is " + myclone.weight);
        System.out.println();

        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
        System.out.println("Weight of mycube is " + mycube.weight);
        System.out.println();
    }
}
```

3.6 Using super to overcome name hiding.

```
class A3 {
    int i;
}

class B3 extends A3 {
    int i;

    B3(int a, int b) {
        super.i = a;
        i = b;
    }

    void show() {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
    }
}

public class UseSuper {

    public static void main(String[] args) {
        B3 subOb = new B3(1, 2);

        subOb.show();
    }
}
```

3.7 Extend BoxWeight to include shipping costs.

```
public class Shipment extends BoxWeight {
    double cost;

    Shipment(Shipment ob) {
        super(ob);
        cost = ob.cost;
    }

    Shipment(double w, double h, double d, double m, double c) {
        super(w, h, d, m);
        cost = c;
    }

    Shipment() {
        super();
        cost = -1;
    }

    Shipment(double len, double m, double c) {
        super(len, m);
        cost = c;
    }
}
```

```
public class DemoShipment {

    public static void main(String[] args) {
        Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
        Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);
        double vol;

        vol = shipment1.volume();
        System.out.println("Volume of shipment1 is " + vol);
        System.out.println("Weight of shipment1 is " + shipment1.weight);
        System.out.println("Shipping cost: $ " + shipment1.cost);
        System.out.println();

        vol = shipment2.volume();
        System.out.println("Volume of shipment2 is " + vol);
        System.out.println("Weight of shipment2 is " + shipment2.weight);
        System.out.println("Shipping cost: $ " + shipment2.cost);
    }
}
```

3.8 Demonstrate when constructors are executed.

```
class A4 {
    A4() {
        System.out.println("Inside A's constructor.");
    }
}

class B4 extends A4 {
    B4() {
        System.out.println("Inside B's constructor.");
    }
}

class C4 extends B4 {
    C4() {
        System.out.println("Inside C's constructor.");
    }
}

public class CallingCons {

    public static void main(String[] args) {
        C4 c = new C4();
    }
}
Footer
```

3.9 Method overriding.

```
class A5 {
    int i, j;

    A5(int a, int b) {
        i = a;
        j = b;
    }

    void show() {
        System.out.println("i and j: " + i + " " + j);
    }
}

class B5 extends A5 {
    int k;

    B5(int a, int b, int c) {
        super(a, b);
        k = c;
    }

    void show() {
        System.out.println("k: " + k);
    }
}

public class Override {

    public static void main(String[] args) {
        B5 subOb = new B5(1, 2, 3);

        subOb.show();
    }
}
Footer
```

3.10 Dynamic method dispatch

```
class A7 {
    void callme() {
        System.out.println("Inside A's callme method");
    }
}

class B7 extends A7 {
    void callme() {
        System.out.println("Inside B's callme method");
    }
}

class C7 extends A7 {
    void callme() {
        System.out.println("Inside C's callme method");
    }
}

public class Dispatch {

    public static void main(String[] args) {
        A7 a = new A7();
        B7 b = new B7();
        C7 c = new C7();

        A7 r;

        r = a;
        r.callme();

        r = b;
        r.callme();

        r = c;
        r.callme();
    }
}
```

3.11 Method with differing type signatures are overloaded - not overridden.

```
class A6 {
    int i, j;

    A6(int a, int b) {
        i = a;
        j = b;
    }

    void show() {
        System.out.println("i and j: " + i + " " + j);
    }
}

class B6 extends A6 {
    int k;

    B6(int a, int b, int c) {
        super(a, b);
        k = c;
    }

    void show(String msg) {
        System.out.println(msg + k);
    }
}

public class Overloaded {

    public static void main(String[] args) {
        B6 subOb = new B6(1, 2, 3);

        subOb.show("This is k: ");
        subOb.show();
    }
}
```

3.12 Dynamic method dispatch

```
class A7 {
    void callme() {
        System.out.println("Inside A's callme method");
    }
}

class B7 extends A7 {
    void callme() {
        System.out.println("Inside B's callme method");
    }
}

class C7 extends A7 {
    void callme() {
        System.out.println("Inside C's callme method");
    }
}

public class Dispatch {

    public static void main(String[] args) {
        A7 a = new A7();
        B7 b = new B7();
        C7 c = new C7();

        A7 r;

        r = a;
        r.callme();

        r = b;
        r.callme();

        r = c;
        r.callme();
    }
}
```

3.13 Using run-time polymorphism

```
class Figure {
    double dim1;
    double dim2;

    Figure(double a, double b) {
        dim1 = a;
        dim2 = b;
    }

    double area() {
        System.out.println("Area for Figure is undefined.");
        return 0;
    }
}

class Rectangle extends Figure {
    Rectangle(double a, double b) {
        super(a, b);
    }

    double area() {
        System.out.println("Inside Area for Rectangle.");
        return dim1 * dim2;
    }
}

class Triangle extends Figure {
    Triangle(double a, double b) {
        super(a, b);
    }

    double area() {
        System.out.println("Inside Area for Triangle.");
        return dim1 * dim2 / 2;
    }
}

class FindAreas {

    public static void main(String[] args) {
        Figure f = new Figure(10, 10);
        Rectangle r = new Rectangle(9, 5);

        Triangle t = new Triangle(10, 8);
        Figure figref;

        figref = r;
```

```
        System.out.println("Area is " + figref.area());

        figref = t;
        System.out.println("Area is " + figref.area());

        figref = f;
        System.out.println("Area is " + figref.area());
    }
}
Footer
```

3.14 A simple demonstration of abstract.

```
abstract class A8 {
    abstract void callme();

    void callmetoo() {
        System.out.println("This is a concrete method.");
    }
}

class B8 extends A8 {
    void callme() {
        System.out.println("B's implementation of callme.");
    }
}

class AbstractDemo {

    public static void main(String[] args) {
        B8 b = new B8();

        b.callme();
        b.callmetoo();
    }
}
```

3.15 Using abstract methods and classes.

```
abstract class Figure2 {
    double dim1;
    double dim2;

    Figure2(double a, double b) {
        dim1 = a;
        dim2 = b;
    }

    abstract double area();
}

class Rectangle2 extends Figure2 {
    Rectangle2(double a, double b) {
        super(a, b);
    }

    double area() {
        System.out.println("Inside Area for Rectangle.");
```

```
        return dim1 * dim2;
    }
}

class Triangle2 extends Figure2 {
    Triangle2(double a, double b) {
        super(a, b);
    }

    double area() {
        System.out.println("Inside Area for Triangle.");
        return dim1 * dim2 / 2;
    }
}

public class AbstractAreas {

    public static void main(String[] args) {
        // Figure2 f = new Figure(10, 10);
        Rectangle2 r = new Rectangle2(9, 5);

        Triangle2 t = new Triangle2(10, 8);
        Figure2 figref;

        figref = r;
        System.out.println("Area is " + figref.area());

        figref = t;
        System.out.println("Area is " + figref.area());
    }
}
```

3.16 Return from values from derived slasses

```
class MyClass {
}

class FirstDerivedClass extends MyClass {
    int x;
}

class SecondDerivedClass extends FirstDerivedClass {
    int y;
}

public class TypeInferenceAndInheritance {

    static MyClass getObj(int which) {
        switch (which) {
        case 0:
            return new MyClass();
        case 1:
            return new FirstDerivedClass();
        default:
            return new SecondDerivedClass();
        }
    }

    public static void main(String[] args) {
        var mc = getObj(0);

        var mc2 = getObj(1);

        var mc3 = getObj(2);

        // mc2.x = 10;
        // mc3.y = 10;
    }
}
```

4 Packages

- Defining packages
- Creating packages
- Importing and accessing packages

4.1 A simple package

```
package mypack;

public class Balance {
    String name;
    double bal;

    public Balance(String n, double b) {
        name = n;
        bal = b;
    }

    public void show() {
        if (bal < 0) {
            System.out.print("--> ");
        }
        System.out.println(name + ": $" + bal);
    }
}
```

```
package mypack;

public class AccountBalance {

    public static void main(String[] args) {
        Balance current[] = new Balance[3];

        current[0] = new Balance("K. J. Fielding", 123.23);
        current[1] = new Balance("Will Tell", 157.02);
        current[2] = new Balance("Tom Jackson", -12.33);

        for (int i = 0; i < 3; i++)
            current[i].show();
    }
}
```

4.2 Package Access Examples

```
package p1;

public class Protection {
    int n = 1;
    private int n_pri = 2;
    protected int n_pro = 3;
    public int n_pub = 4;

    public Protection() {
        System.out.println("base constructor");
        System.out.println("n = " + n);
        System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

```
package p1;

class Derived extends Protection {
    Derived() {
        System.out.println("derived constructor");
        System.out.println("n = " + n);
        // System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

```
package p1;

class SamePackage {
    SamePackage() {
        Protection p = new Protection();
        System.out.println("same package constructor");
        System.out.println("n = " + p.n);

        // System.out.println("n_pri = " + p.n_pri);
        System.out.println("n_pro = " + p.n_pro);
        System.out.println("n_pub = " + p.n_pub);
    }
}
```

```
package p1;

public class Demo {

    public static void main(String[] args) {
        Protection ob1 = new Protection();
        Derived ob2 = new Derived();
        SamePackage ob3 = new SamePackage();
    }
}
```

```java
package p2;

import p1.Protection;

class OtherPackage {
    OtherPackage() {
        p1.Protection p = new Protection();
        System.out.println("other package constructor");

        // System.out.println("n = " + p.n);

        // System.out.println("n_pri = " + p.n_pri);
        // System.out.println("n_pro = " + p.n_pro);
        System.out.println("n_pub = " + p.n_pub);
    }
}
```

```java
package p2;

import p1.Protection;

class Protection2 extends Protection {
    Protection2() {
        System.out.println("derived other package constructor");

        // System.out.println("n = " + n);
        // System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

```java
package p2;

public class Demo {

    public static void main(String[] args) {
        Protection2 ob1 = new Protection2();
        OtherPackage pb2 = new OtherPackage();
    }
}
```

4.3 Package Import from mypack

```
import mypack.*;

public class TestBalance {

    public static void main(String[] args) {
        Balance test = new Balance("J. J. Jaspers", 99.88);

        test.show();
    }
}
```

5 Interfaces

- Interfaces: Defining, implementing and extending interfaces
- Variables in interfaces,
- Default Method in Interface
- Static Method in interface
- Abstract Classes vs Interfaces.

5.1 Simple interface example

```
public interface Callback {

    void callback(int param);
}
```

```
public class Client implements Callback {

    public void callback(int p) {
        System.out.println("callback called with " + p);
    }

    void nonIfaceMeth() {
        System.out.println("Classes that implement interfaces " + "may
            also define other members, too.");
    }
}
```

```
public class TestIface {

    public static void main(String[] args) {
        Callback c = new Client();
        c.callback(42);
    }
}
```

5.2 Another implementation of Callback.

```
public class AnotherClient implements Callback {

    public void callback(int p) {
        System.out.println("Another version of callback");
        System.out.println("p squared is " + (p * p));
    }
}
```

```
public class TestIface2 {

    public static void main(String[] args) {
        Callback c = new Client();
        AnotherClient ob = new AnotherClient();

        c.callback(42);

        c = ob;
        c.callback(42);
    }
}
```

5.3 A nested interface example

```
class A {
    public interface NestedIF {
        boolean isNotNegative(int x);
    }
}

class B implements A.NestedIF {
    public boolean isNotNegative(int x) {
        return x < 0 ? false : true;
    }
}

class NestedIFDemo {
    public static void main(String[] args) {
        A.NestedIF nif = new B();

        if (nif.isNotNegative(10)) {
            System.out.println("10 is not negative");
        }

        if (nif.isNotNegative(-12)) {
```

```
            System.out.println("this won't be displayed");
        }
    }
}
```

5.4 Stack Interface Example

```
public interface IntStack {

    void push(int item);

    int pop();
}
```

```
public class FixedStack implements IntStack {

    private int stck[];
    private int tos;

    FixedStack(int size) {
        stck = new int[size];
        tos = -1;
    }

    public void push(int item) {
        if (tos == stck.length - 1) {
            System.out.println("Stack is full.");
        } else {
            stck[++tos] = item;
        }
    }

    public int pop() {
        if (tos < 0) {
            System.out.println("Stack underflow.");

            return 0;
        } else {
            return stck[tos--];
        }
    }
}
```

```
public class IFTest {

    public static void main(String[] args) {
        FixedStack mystack1 = new FixedStack(5);
        FixedStack mystack2 = new FixedStack(8);

        for (int i = 0; i < 5; i++)
            mystack1.push(i);
        for (int i = 0; i < 8; i++)
            mystack2.push(i);

        System.out.println("Stack in mystack1:");
        for (int i = 0; i < 5; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stack in mystack2:");
        for (int i = 0; i < 8; i++)
            System.out.println(mystack2.pop());
    }
}
```

5.5 Dynamic Stack using IntStack , Implement a "growable" stack.

```
public class DynStack implements IntStack {
    private int stck[];
    private int tos;

    DynStack(int size) {
        stck = new int[size];
        tos = -1;
    }

    public void push(int item) {
        if (tos == stck.length - 1) {
            int temp[] = new int[stck.length * 2];
            for (int i = 0; i < stck.length; i++)
                temp[i] = stck[i];
            stck = temp;
            stck[++tos] = item;
        } else {
            stck[++tos] = item;
        }
    }

    public int pop() {
        if (tos < 0) {
```

```
            System.out.println("Stack underflow.");

            return 0;
        } else {
            return stck[tos--];
        }
    }
}
```

```
public class IfTest2 {

    public static void main(String[] args) {
        DynStack mystack1 = new DynStack(5);
        DynStack mystack2 = new DynStack(8);

        for (int i = 0; i < 12; i++)
            mystack1.push(i);
        for (int i = 0; i < 20; i++)
            mystack2.push(i);

        System.out.println("Stack in mystack1:");
        for (int i = 0; i < 12; i++)
            System.out.println(mystack1.pop());

        System.out.println("Stack in mystack2:");
        for (int i = 0; i < 20; i++)
            System.out.println(mystack2.pop());
    }
}
```

5.6 Create an interface variable and access stacks through it

```
public class IFTest3 {

    public static void main(String[] args) {
        IntStack mystack;
        DynStack ds = new DynStack(5);
        FixedStack fs = new FixedStack(8);

        mystack = ds;
        for (int i = 0; i < 12; i++)
            mystack.push(i);

        mystack = fs;
        for (int i = 0; i < 8; i++)
```

```
            mystack.push(i);

        mystack = ds;
        System.out.println("Values in dynamic stack:");
        for (int i = 0; i < 12; i++)
            System.out.println(mystack.pop());

        mystack = fs;
        System.out.println("Values in fixed stack:");
        for (int i = 0; i < 8; i++)
            System.out.println(mystack.pop());
    }
}
```

5.7 Shared constants in the Iterface

```
import java.util.Random;

interface SharedConstants {
    int NO = 0;
    int YES = 1;
    int MAYBE = 2;
    int LATER = 3;
    int SOON = 4;
    int NEVER = 5;
}

class Question implements SharedConstants {
    Random rand = new Random();

    int ask() {
        int prob = (int) (100 * rand.nextDouble());
        if (prob < 30) {
            return NO;
        } else if (prob < 60) {
            return YES;
        } else if (prob < 75) {
            return LATER;
        } else if (prob < 98) {
            return SOON;
        } else {
            return NEVER;
        }
    }
}

class AskMe implements SharedConstants {
    static void answer(int result) {
```

```
        switch (result) {
        case NO:
            System.out.println("No");
            break;
        case YES:
            System.out.println("Yes");
            break;
        case MAYBE:
            System.out.println("Maybe");
            break;
        case LATER:
            System.out.println("Later");
            break;
        case SOON:
            System.out.println("Soon");
            break;
        case NEVER:
            System.out.println("Never");
            break;
        }
    }

    public static void main(String[] args) {
        Question q = new Question();

        answer(q.ask());
        answer(q.ask());
        answer(q.ask());
        answer(q.ask());
    }
}
```

5.8 Interface Extended

```
interface A2 {
    void meth1();

    void meth2();
}

interface B2 extends A2 {
    void meth3();
}

class MyClass implements B2 {
    public void meth1() {
        System.out.println("Implement meth1().");
    }

    public void meth2() {
        System.out.println("Implement meth2().");
    }

    public void meth3() {
        System.out.println("Implement meth3().");
    }
}

public class IFExtend {

    public static void main(String[] args) {
        MyClass ob = new MyClass();

        ob.meth1();
        ob.meth2();
        ob.meth3();
    }
}
```

5.9 Use the default method.

```
public interface MyIF {

    int getNumber();

    default String getString() {
        return "Default String";
    }
}
```

```
class MyIFImp implements MyIF {
    public int getNumber() {
        return 100;
    }
}
```

```
public class DefaultMethodDemo {

    public static void main(String[] args) {
        MyIFImp obj = new MyIFImp();

        System.out.println(obj.getNumber());

        System.out.println(obj.getString());
    }
}
```

6 Exception Handling

- Exception-Handling Fundamentals
- Exception Types
- Exception class Hierarchy
- Using try and catch
- Multiple catch Clauses
- Nested try Statements
- throw, throws and finally
- Java's Built-in Exceptions
- Creating Your Own Exception Subclasses

6.1 Divide-by-zero error

```
class Exc0 {
    public static void main(String[] args) {
        int d = 0;
        int a = 42 / d;
    }
}
```

6.2 Divide-by-zero error using method

```
class Exc1 {
    static void subroutine() {
        int d = 0;
        int a = 10 / d;
    }

    public static void main(String[] args) {
        Exc1.subroutine();
    }
}
```

6.3 Division by zero exception

```
class Exc2 {
    public static void main(String[] args) {
        int d, a;


            d = 0;

        try {
            a = 42 / d;
            System.out.println("This will not be printed.");
        } catch (ArithmeticException e) {
            System.out.println("Division by zero.");
        }

        System.out.println("After catch statement.");

    }
}
```

6.4 ArithmeticException

```
import java.util.Random;

class HandleError {
    public static void main(String[] args) {
        int a = 0, b = 0, c = 0;

        Random r = new Random();

        for (int i = 0; i < 32000; i++) {
            try {
                b = r.nextInt();
                c = r.nextInt();
                a = 12345 / (b / c);
            } catch (ArithmeticException e) {
                System.out.println("Exception: " + e);
                a = 0;
            }
            System.out.println("a: " + a);
        }
    }
}
```

6.5 MultipleCatch Divide by Zero

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

        try {
            int a = args.length;
            System.out.println("a = " + a);
            int b = 42 / a;
            int[] c = {1};
            c[42] = 99;

        } catch (ArithmeticException e) {
            System.out.println("Divide by 0: " + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index oob: " + e);
        }

        System.out.println("After try/catch blocks.");
    }
}
```

6.6 SuperSub Catch

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

        try {
            int a = 0;
            int b = 42 / a;

        } catch (ArithmeticException e) {
            System.out.println("This is never reached.");
        } catch (Exception e) {
            System.out.println("Generic Exception catch.");
        }

    }
}
```

6.7 NestedTry Array out of bound

```
class NestTry {
    public static void main(String[] args) {
        try {
            int a = args.length;
            int b = 42 / a;

            System.out.println("a = " + a);

            try {

                if (a == 1) {
                    a = a / (a - a);
                }

                if (a == 2) {
                    int c[] = {1};
                    c[42] = 99;
                }

            }  catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Array index out-of-bounds: " + e);
            }

        } catch (ArithmeticException e) {
            System.out.println("Divide by 0: " + e);
        }
    }
}
```

6.8 MethNestTry

```
class MethNestTry {
    static void nestTry(int a) {
        try {

            if (a == 1) {
                a = a / (a - a);
            }

            if (a == 2) {
                int c[] = {1};
                c[42] = 99;
            }

        }  catch (ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println("Array index out-of-bounds: " + e);
        }
    }

    public static void main(String[] args) {
        try {
            int a = args.length;
            int b = 42 / a;

            System.out.println("a = " + a);

            MethNestTry.nestTry(a);

        } catch (ArithmeticException e) {
            System.out.println("Divide by 0: " + e);
        }
    }
}
```

6.9 ThrowDemo

```
class ThrowDemo {
    static void demoProc() {
        try {
            throw new NullPointerException("Demo");
        } catch (NullPointerException e) {
            System.out.println("Caught inside demoProc.");
            throw e;
        }
    }

    public static void main(String[] args) {
        try {
            demoProc();
        } catch (NullPointerException e) {
            System.out.println("Recaught: " + e);
        }
    }
}
```

6.10 ThrowsDemo

```
class ThrowsDemo {
    static void throwOne() throws IllegalAccessException {
        System.out.println("Inside throwOne.");
        throw new IllegalAccessException("demo");
    }

    public static void main(String[] args) {
        try {
            throwOne();
        } catch (IllegalAccessException e) {
            System.out.println("Caught " + e);
        }
    }
}
Footer
```

6.11 FinallyDemo

```
class FinallyDemo {


    static void procA() {
        try {
            System.out.println("inside procA");
            throw new RuntimeException("demo");
        } finally {
            System.out.println("procA's finally");
        }
    }

    static void procB() {
        try {
            System.out.println("inside procB");
            return;
        } finally {
            System.out.println("procB's finally");
        }
    }

    static void procC() {
        try {
            System.out.println("inside procC");
        } finally {
            System.out.println("procC's finally");
        }
```

```
    }

    public static void main(String[] args) {

        try {
            procA();
        } catch (Exception e) {
            System.out.println("Exception caught");
        }

        procB();
        procC();
    }
}
```

6.12 ChainExcDemo

```
class ChainExcDemo {
    static void demoproc() {

        NullPointerException e = new NullPointerException("top layer");

        e.initCause(new ArithmeticException("cause"));

        throw e;
    }

    public static void main(String[] args) {

        try {
            demoproc();
        } catch (NullPointerException e) {

            System.out.println("Caught: " + e);

            System.out.println("Original cause: " + e.getCause());
        }

    }
}
```

6.13 MultiCatch

```
class MultiCatch {

    public static void main(String[] args) {

        int a = 10, b = 0;
        int vals[] = {1, 2, 3};

        try {
            int result = a / b;

            // vals[10] = 19;

        } catch (ArithmeticException | ArrayIndexOutOfBoundsException e)
            {
            System.out.println("Exception caught: " + e);
        }

        System.out.println("After multi-catch.");

    }

}
```

7 Multi-Threading

- The Java Thread Model and Thread Life Cycle
- Thread Priorities
- Creating a Thread
- Implementing Runnable
- Extending Thread
- Creating Multiple Threads
- Synchronization: Using Synchronized Methods
- The synchronized Statement

7.1 Thread Demo

```
class NewThread implements Runnable {

    Thread t;

    NewThread() {
        t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
    }

    @Override
    public void run() {

        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }

        System.out.println("Exiting child thread.");
    }
}

class ThreadDemo {
    public static void main(String[] args) {
        NewThread nt = new NewThread();

        nt.t.start();
```

```
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("Main thread exiting.");
    }
}
```

7.2 CurrentThreadDemo

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

        Thread t = Thread.currentThread();

        System.out.println("Current thread: " + t);
        System.out.println(t.getName());

        t.setName("My Thread");
        System.out.println("After name change: " + t);
        System.out.println(t.getName());

        try {
            for (int n = 6; n > 0; n--) {
                System.out.println(n);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }
    }
}
```

7.3 Thread Demo

```
class NewThread2 extends Thread {

    NewThread2() {
        super("Demo Thread");
        System.out.println("Child thread: " + this);
    }

    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child Thread: " + i);
                Thread.sleep(500);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }

        System.out.println("Exiting child thread.");
    }
}

class ExtendThread {

    public static void main(String[] args) {
        NewThread2 nt = new NewThread2();

        nt.start();

        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Main thread exiting.");
    }
}
```

7.4 MultiThread Demo

```
class NewThread3 implements Runnable {

    String name;
    Thread t;

    NewThread3(String threadName) {
        name = threadName;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
    }

    @Override
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted.");
        }
        System.out.println(name + " exiting.");
    }

}

class MultiThreadDemo {
    public static void main(String[] args) {
        NewThread3 nt1 = new NewThread3("One");
        NewThread3 nt2 = new NewThread3("Two");
        NewThread3 nt3 = new NewThread3("Three");

        nt1.t.start();
        nt2.t.start();
        nt3.t.start();

        try {
            Thread.sleep(10000);

        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }
        System.out.println("Exiting main thread.");
    }
}
```

7.5 Thread Join Demo

```
class NewThread implements Runnable {

    String name;
    Thread t;

    NewThread(String name) {
        this.name = name;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
    }

    @Override
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted.");
        }
        System.out.println(name + " exiting.");
    }
}

class DemoJoin {
    public static void main(String[] args) {
        NewThread nt1 = new NewThread("One");
        NewThread nt2 = new NewThread("Two");
        NewThread nt3 = new NewThread("Three");

        // Start the threads.
        nt1.t.start();
        nt2.t.start();
        nt3.t.start();

        System.out.println("Thread One is alive: " + nt1.t.isAlive());
        System.out.println("Thread Two is alive: " + nt2.t.isAlive());
        System.out.println("Thread Three is alive: " + nt3.t.isAlive());

        // Wait for threads to finish.
        try {
            System.out.println("Waiting for threads to finish.");
            nt1.t.join();
            nt2.t.join();
            nt3.t.join();
        } catch (InterruptedException e) {
```

```
            System.out.println("Main thread interrupted.");
        }

        System.out.println("Thread One is alive: " + nt1.t.isAlive());
        System.out.println("Thread Two is alive: " + nt2.t.isAlive());
        System.out.println("Thread Three is alive: " + nt3.t.isAlive());

        System.out.println("Main thread exiting.");
    }
}
```

7.6 Thread Synchronization

```
class Callme {
    void call(String msg) {
        System.out.print("[" + msg);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Interrupted");
        }
        System.out.println("]");
    }
}

class Caller extends Thread {
    String msg;
    Callme target;

    public Caller(Callme target, String msg) {
        super();
        this.target = target;
        this.msg = msg;
    }

    @Override
    public void run() {
        synchronized(target) {
            target.call(msg);
        }
    }
}

public class Synch1 {
    public static void main(String[] args) {
        Callme target = new Callme();
        Caller obj1 = new Caller(target, "Hello");
        Caller obj2 = new Caller(target, "Synchronized");
```

```
        Caller obj3 = new Caller(target, "World");

        obj1.start();
        obj2.start();
        obj3.start();

        try {
            obj1.join();
            obj2.join();
            obj3.join();
        } catch (InterruptedException e) {
            System.out.println("Interrupted");
        }
    }
}
```

7.7 Producer Consumer Problem

```
class Q {
    int n;

    synchronized int get() {
        System.out.println("Got: " + n);
        return n;
    }

    synchronized void put(int n) {
        this.n = n;
        System.out.println("Put: " + n);
    }
}

class Producer implements Runnable {
    Q q;
    Thread t;

    Producer(Q q) {
        this.q = q;
        t = new Thread(this, "Producer");
    }

    public void run() {
        int i = 0;

        while (true) {
            q.put(i++);
        }
    }
```

```
}

class Consumer implements Runnable {
    Q q;
    Thread t;

    Consumer(Q q) {
        this.q = q;
        t = new Thread(this, "Consumer");
    }

    public void run() {
        while (true) {
            q.get();
        }
    }
}

public class PC {
    public static void main(String[] args) {
        Q q = new Q();
        Producer p = new Producer(q);
        Consumer c = new Consumer(q);

        p.t.start();
        c.t.start();

        System.out.println("Press Control-F2 to stop.");
    }
}
```

7.8 Producer Consumer Problem Fixed

```
class Q {
    int n;
    boolean valueSet = false;

    synchronized int get() {
        while (!valueSet) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("InterruptedException caught");
            }
        }
        System.out.println("Got: " + n);
        valueSet = false;
        notify();
```

```
        return n;
    }

    synchronized void put(int n) {
        while (valueSet) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("InterruptedException caught");
            }
        }

        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        notify();
    }
}

class Producer implements Runnable {
    Q q;
    Thread t;

    Producer(Q q) {
        this.q = q;
        t = new Thread(this, "Producer");
    }

    @Override
    public void run() {
        int i = 0;
        while (true) {
            q.put(i++);
        }
    }
}

class Consumer implements Runnable {
    Q q;
    Thread t;

    Consumer(Q q) {
        this.q = q;
        t = new Thread(this, "Consumer");
    }

    @Override
    public void run() {
        while (true) {
            q.get();
```

```
        }
    }
}

public class PCFixed {
    public static void main(String[] args) {
        Q q= new Q();
        Producer p = new Producer(q);
        Consumer c = new Consumer(q);

        p.t.start();
        c.t.start();

        System.out.println("Press Control-F2 to stop.");
    }
}
```

7.9 Deadlock

```
class A {
    synchronized void foo(B b) {
        String name = Thread.currentThread().getName();

        System.out.println(name + " entered A.foo");


        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("A Interrupted");
        }



        System.out.println(name + " trying to call B.last()");
        b.lastB(); // MainThread owns the monitor on a and is waiting
            for the monitor on b
    }

    synchronized void lastA() {
        System.out.println("Inside A.last");
    }
}

class B {
    synchronized void bar(A a) {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered B.bar");
```

```
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("B Interrupted");
        }



        System.out.println(name + " trying to call A.last()");
        a.lastA(); // RacingThread owns the monitor on b and is waiting
            for the monitor on a
    }

    synchronized void lastB() {
        System.out.println("Inside B.last");
    }
}

public class Deadlock implements Runnable {
    A a = new A();
    B b = new B();
    Thread t;

    Deadlock() {
        Thread.currentThread().setName("MainThread");
        t = new Thread(this, "RacingThread");
    }

    void deadlockStart() {
        t.start();
        a.foo(b);
        System.out.println("Back in main thread");
    }

    public void run() {
        b.bar(a);
        System.out.println("Back in other thread");
    }

    public static void main(String[] args) {
        Deadlock d1 = new Deadlock();

        d1.deadlockStart();

    }
}
```

7.10 Thread States Demo

```
class NewThread implements Runnable {

    String name;
    Thread t;
    boolean suspendFlag;

    NewThread(String name) {
        this.name = name;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        suspendFlag = false;
    }

    @Override
    public void run() {
        try {
            for (int i = 15; i > 0; i--) {
                System.out.println(name + ": " + i);
                Thread.sleep(200);
                synchronized (this) {
                    while (suspendFlag) {
                        wait();
                    }
                }
            }
        } catch (InterruptedException e) {
            System.out.println(name + " interrupted.");
        }
        System.out.println(name + " exiting.");
    }

    synchronized void mySuspend() {
        suspendFlag = true;
    }

    synchronized void myResume() {
        suspendFlag = false;
        notify();
    }
}

public class SuspendResume {
    public static void main(String[] args) {
        NewThread obj1 = new NewThread("One");
        NewThread obj2 = new NewThread("Two");

        System.out.println(obj1.t.getState());
```

```
        obj1.t.start();
        obj2.t.start();

        try {
            Thread.sleep(1000);

            obj1.mySuspend();
            System.out.println("Suspending thread One");
            Thread.sleep(1000);
            obj1.myResume();
            System.out.println("Resuming thread One");

            obj2.mySuspend();
            System.out.println(obj2.t.getState());
            System.out.println("Suspending thread Two");
            System.out.println(obj2.t.getState());
            Thread.sleep(1000);
            obj2.myResume();
            System.out.println("Resuming thread Two");
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }
        System.out.println(Thread.currentThread().getState());
        try {
            System.out.println("Waiting for threads to finish.");
            obj1.t.join();
            obj2.t.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted");
        }

        Thread.State ts = obj2.t.getState();
        System.out.println(obj1.t.getState());
        System.out.println(ts);

        if (ts == Thread.State.TERMINATED) {
            System.out.println("Child thread 2 is terminated.");
        }

        System.out.println("Main thread exiting.");
    }
}
```

7.11 ThreadDemo

```
class NewThread implements Runnable {

    Thread t;

    NewThread() {
        t = new Thread(this, "Demo Thread");
        System.out.println("Child thread: " + t);
    }

    @Override
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Child thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Child interrupted.");
        }
        System.out.println("Exiting child thread.");
    }

    public static NewThread createAndStart() {
        NewThread myThrd = new NewThread();
        myThrd.t.start();
        return myThrd;
    }
}

public class ThreadDemo {
    public static void main(String[] args) {
        NewThread nt = NewThread.createAndStart();

        try {
            for (int i = 5; i > 0; i--) {
                System.out.println("Main Thread: " + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            System.out.println("Main thread exiting.");
        }
    }
}
```

7.12 BufferedReader Example

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class BRRead {
    public static void main(String[] args) throws IOException {

        char c;
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

        System.out.println("Enter characters, 'q' to quit.");

        do {
            c = (char) br.read();
            System.out.println(c);
        } while (c != 'q');

    }
}
```

7.13 BufferedReader Read Lines

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class BRReadLines {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

        String str;
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'stop' to quit.");

        do {
            str = br.readLine();
            System.out.println(str);
        } while (!str.equals("stop"));
    }
}
```

7.14 TinyEditor

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

class TinyEdit {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
            InputStreamReader(System.in));

        String[] str = new String[100];
        System.out.println("Enter lines of text.");
        System.out.println("Enter 'stop' to quit.");
        for (int i = 0; i < 100; i++) {
            str[i] = br.readLine();
            if (str[i].equals("stop")) {
                break;
            }
        }
        System.out.println("\nHere is your file:");
        for (int i = 0; i < 100; i++) {
            if (str[i].equals("stop")) {
                break;
            }
            System.out.println(str[i]);
        }
    }

}
Footer
```

7.15 WriteDemo

```
class WriteDemo {

    public static void main(String[] args) {

        int b;

        b = 'A';
        System.out.write(b);
        System.out.write('\n');

    }
```

```
}
```

7.16 PrintWriterDemo

```
import java.io.PrintWriter;

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

        PrintWriter pw = new PrintWriter(System.out, true);

        pw.println("This is a string");
        int i = -7;
        pw.println(i);
        double d = 4.5e-7;
        pw.println(d);

    }
}
```

7.17 ShowFile

```
/* Display a text file.
   To use this program, specify the name of the file that you want to
        see.
   For example, to see a file called TEST.TXT, use the following command
        line.
   java showFile TEST.TXT
 */

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;


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

        int i;
        FileInputStream fin;

        // First, confirm that a filename has been specified.
        if (args.length != 1) {
            System.out.println("Usage: ShowFile filename");
            return;
```

```
        }

        // Attempt to open the file.
        try {
            fin = new FileInputStream(args[0]);
        } catch (FileNotFoundException e) {
            System.out.println("Cannot Open File");
            return;
        }

        // At this point, the file is open and can be read.
        // The following reads characters until EOF is encountered.

        try {
            i = fin.read();
            System.out.println(i);
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println();
        System.out.println();

        try {
            do {
                i = fin.read();
                if (i != -1) {
                    System.out.print((char) i);
                }
            } while (i != -1) ;

        } catch (IOException e) {
            System.out.println("Error Reading File");
        }

        // Close the file.
        try {
            fin.close();
        } catch (IOException e) {
            System.out.println("Error Closing File");
        }
    }
}
```

7.18 ShowFile2

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

class ShowFile2 {
    public static void main(String[] args) {

        int i;
        FileInputStream fin = null;

        // First, confirm that a filename has been specified.
        if (args.length != 1) {
            System.out.println("Usage: ShowFile2 filename");
            return;
        }

        // The following code opens a file, reads characters until EOF
            is encountered,
        // and then closes the file via a finally block.
        try {
            fin = new FileInputStream(args[0]);

            do {
                i = fin.read();
                if (i != -1) {
                    System.out.print((char) i);
                }
            } while (i != -1);

        } catch (IOException e) {
            System.out.println("I/O Error: " + e.getClass());
        } finally {
            // Close file in all cases.
            try {
                if (fin != null) {
                    fin.close();
                }
            } catch (IOException e) {
                System.out.println("Error Closing File");
            }

        }
    }
}
```

7.19 CopyFile

```
/*  Copy a file.
    To use this program, specify the name of the source file and the
        destination file.
    For example, to copy a file called FIRST.TXT to a file called
        SECOND.TXT, use the following command line.
    java CopyFile FIRST.TXT SECOND.TXT
 */

import java.io.*;

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

        int i;
        FileInputStream fin = null;
        FileOutputStream fout = null;

        // First, confirm that both files have been specified.
        if (args.length != 2) {
            System.out.println("Usage: CopyFile from to");
            return;
        }

        // Copy a File.
        try {
            // Attempt to open the files.
            fin = new FileInputStream(args[0]);
            fout = new FileOutputStream(args[1]);

            do {
                i = fin.read();
                if (i != -1) {
                    fout.write(i);
                }
            } while (i != -1);

        } catch (IOException e) {
            System.out.println("I/O Error: " + e);
        } finally {
            try {
                if (fin != null) {
                    fin.close();
                }
            } catch (IOException e) {
                System.out.println("Error Closing Input File");
            }
```

```
            try {
                if (fout != null) {
                    fout.close();
                }
            } catch (IOException e) {
                System.out.println("Error Closing Output File");
            }
        }
    }
}
```