//Write a java program based on basic syntactical constructs

```
import java.util.Scanner;

public class SimpleBasics{

// Constant
public static final int MAX = 100;

public static void main(String[] args) {
    // 1) Output
    System.out.println("Hello, Java basics!");

    // 2) Variables and data types
    int a = 10;
    double b = 20.5;
    boolean flag = true;
    char ch = 'X';
    String name = "Shreyash";

    // 3) Arithmetic and assignment
    int sum = a + 5;
    sum += 10; // sum = sum + 10

    // 4) Conditional statements
    if (sum > MAX / 2) {
        System.out.println("Sum is greater than half of
MAX.");
    } else {
        System.out.println("Sum is not greater than half of
MAX.");
    }

    // 5) Switch statement
    int code = 1;
    switch (code) {
        case 1:
            System.out.println("Code is 1");
            break;
        case 2:
            System.out.println("Code is 2");
            break;
        default:
            System.out.println("Code is something else");
    }

    // 6) Loop (for)
    System.out.print("Numbers 1 to 5: ");
    for (int i = 1; i <= 5; i++) {
        System.out.print(i + (i < 5 ? ", " : "\n"));
```

```
    }

    // 7) Array
    int[] nums = {2, 4, 6, 8, 10};
    System.out.print("Array elements: ");
    for (int n : nums) {
        System.out.print(n + " ");
    }
    System.out.println();

    // 8) Method call
    int doubled = doubleValue(7);
    System.out.println("Double of 7 is: " + doubled);

    // 9) Basic input (name)
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter your name: ");
    String userName = scanner.nextLine();
    System.out.println("Nice to meet you, " + userName + "!");
    scanner.close();

    // 10) Simple object usage (inner class)
    Person person = new Person(userName, 25);
    System.out.println("Created person: " + person);
}

// 8) Simple static method
public static int doubleValue(int x) {
    return x * 2;
}

// 10) Simple inner class to demonstrate a custom type
static class Person {
    private String name;
    private int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{name='" + name + "', age=" + age + "}";
    }
}

}
```

Output:

```
Hello, Java basics!
Sum is greater than half of MAX.
Code is 1
Numbers 1 to 5: 1, 2, 3, 4, 5
Array elements: 2 4 6 8 10
Double of 7 is: 14
Enter your name: Shreyash
Nice to meet you, Shreyash!
Created person: Person{name='Shreyash', age=25}
```

// Write a java program to define a class ,method calling

```
import java.util.*; // example import; not required for this
minimal version


public class SingleFileDemo {


// A nested class to illustrate defining a class
    public static class Calculator {
        public int add(int a, int b) { return a + b; }
        public int multiply(int a, int b) { return a * b; }
        public String greet(String name) { return "Hello, " +
name + "!"; }
    }


    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("sum: " + calc.add(5, 7));
        System.out.println("product: " + calc.multiply(3, 4));
        System.out.println("greeting: " +
calc.greet("Shreyash!"));
    }
}
```

Output:

```
sum: 12
product: 12
greeting: Hello, Shreyash!
```

// Write a java program to implement the following concepts;

//1.Function overloading

//2.Constructor of all types

//3.Default parameters

```
public class ItemDemo {

// The main class to demonstrate functionality
public static void main(String[] args) {
    // 1) Constructors: default, parameterized, copy-like,
varargs
    Item defaultItem = new Item();
    System.out.println("Default item: " + defaultItem);

    Item paramItem = new Item("Widget", 9.99);
    System.out.println("Parameterized item: " + paramItem);

    // Copy-like constructor
    Item copied = new Item(paramItem);
    System.out.println("Copied item: " + copied);

    // Varargs constructor (demonstrates default-like
behavior)
    Item withParts = new Item("Gadget", new
String[]{"Battery", "Screen"});
    System.out.println("Item with parts (varargs): " +
withParts);

    // 2) Method overloading
    // computeTotal with two ints
    int totalInt = Item.computeTotal(5, 7);
    System.out.println("Total (int): " + totalInt);

    // computeTotal with double and int
    double totalMixed = Item.computeTotal(4.5, 3);
    System.out.println("Total (double + int): " + totalMixed);

    // computeTotal with three ints
    int totalThree = Item.computeTotal(1, 2, 3);
    System.out.println("Total (three ints): " + totalThree);

    // 3) Default parameters emulation: use overloaded methods
    double price1 = Item.applyDiscount(19.99);
    double price2 = Item.applyDiscount(19.99, 0.15); //
explicit discount
    System.out.println("Discounted price (default 0.10): " +
price1);
```

```
    System.out.println("Discounted price (explicit 0.15): " +
price2);
}

// 1) Class with varied constructors and toString
static class Item {
    private String name;
    private double price;
    private String[] parts;

    // 1a) Default constructor
    public Item() {
        this.name = "Unknown";
        this.price = 0.0;
        this.parts = new String[0];
    }

    // 1b) Parameterized constructor
    public Item(String name, double price) {
        this.name = name;
        this.price = price;
        this.parts = new String[0];
    }

    // 1c) Copy-like constructor
    public Item(Item other) {
        this.name = other.name;
        this.price = other.price;
        this.parts = other.parts.clone();
    }

    // 1d) Varargs constructor for parts (simulating more
flexible defaults)
    public Item(String name, String... parts) {
        this.name = name;
        // if parts provided, price set based on length for
demo
        this.price = 5.0 + parts.length * 2.5;
        this.parts = parts != null ? parts.clone() : new
String[0];
    }

    // 3) Emulated default parameter via overloading
    // Default discount of 0.10 if not provided
    public static double applyDiscount(double price) {
        return applyDiscount(price, 0.10);
    }

    // Overloaded method with explicit discount rate
    public static double applyDiscount(double price, double
discountRate) {
```

```
        return price * (1.0 - discountRate);
    }

    // 2) Overloaded methods (function overloading)
    // computeTotal with two ints
    public static int computeTotal(int a, int b) {
        return a + b;
    }

    // computeTotal with double and int
    public static double computeTotal(double a, int b) {
        return a + b;
    }

    // computeTotal with three ints
    public static int computeTotal(int a, int b, int c) {
        return a + b + c;
    }

    @Override
    public String toString() {
        String partsStr = (parts.length > 0) ? String.join(",
", parts) : "none";
        return "Item{name='" + name + "', price=" + price + ",
parts=[" + partsStr + "]}";
    }
}

}
```

Output:

```
Default item: Item{name='Unknown', price=0.0,
parts=[none]}

Parameterized item: Item{name='Widget', price=9.99,
parts=[]}

Copied item: Item{name='Widget', price=9.99,
parts=[]}

Item with parts (varargs): Item{name='Gadget',
price=5.0, parts=[Battery, Screen]}

Total (int): 12

Total (double + int): 7.5

Total (three ints): 6

Discounted price (default 0.10): 17.991

Discounted price (explicit 0.15): 16.9935
```

// Write a java program for multilevel and hierarchical inheritance

```
public class MultiHierarchyDemo {

public static void main(String[] args) {
    // Multilevel inheritance: A -> B -> C
    A a = new A();
    B b = new B();
    C c = new C();

    System.out.println("Multilevel Inheritance:");
    a.baseMethod();        // defined in A
    b.baseMethod();        // inherited from A
    b.subMethod();         // defined in B
    c.baseMethod();        // inherited from A
    c.subMethod();         // inherited from B
    c.subSubMethod();      // defined in C

    System.out.println();

    // Hierarchical inheritance: B is the base for D and E
    D d = new D();
    E e = new E();

    System.out.println("Hierarchical Inheritance:");
    d.baseMethod();        // from A via B (A -> B -> D)
    d.overridden();        // D overrides baseMethod
(demonstration)
    e.baseMethod();        // from A via B (A -> B -> E)
    e.subMethod();         // from B
}

// Base class
static class A {
    void baseMethod() {
        System.out.println("A: baseMethod");
    }
}

// Level 1 in multilevel chain
static class B extends A {
    void subMethod() {
        System.out.println("B: subMethod");
    }
}

// Level 2 in multilevel chain
static class C extends B {
    void subSubMethod() {
        System.out.println("C: subSubMethod");
    }
```

```
}

// Hierarchical inheritance: D and E both extend B (sharing A
via B)
static class D extends B {
    // Override to show polymorphic behavior
    @Override
    void baseMethod() {
        System.out.println("D: baseMethod (overridden)");
    }

    // New method specific to D
    void overridden() {
        System.out.println("D: overridden (specific to D)");
    }
}

static class E extends B {
    // Can inherit baseMethod and subMethod from B (and A)
    void subMethod() {
        System.out.println("E: subMethod (inherited from B)");
    }
}}
```

Output:

```
Multilevel Inheritance:

A: baseMethod

B: baseMethod

B: subMethod

A: baseMethod

B: subMethod

C: subSubMethod

Hierarchical Inheritance:

D: baseMethod (overridden)

D: overridden (specific to D)

A: baseMethod

B: subMethod
```

// Write a java program to implement multiple inheritance

```
public class MultipleInheritanceDemo {

public static void main(String[] args) {
    // Using interface-based mixin approach
    Robot android = new AndroidRobot();
    android.move();
    android.speak();

    // Using composition to reuse behavior from multiple
components
    Car car = new Car(new Engine(), new Transmission());
    System.out.println("Car status:");
    car.start();
    car.shiftGear(3);
    car.engineInfo();
}

// Interface-based "multiple inheritance" of behavior
interface Movable {
    void move();
}

interface Speakable {
    void speak();
}

// Concrete class implementing multiple interfaces
static class AndroidRobot implements Movable, Speakable {
    @Override
    public void move() {
        System.out.println("AndroidRobot is moving: walking
humanoid steps.");
    }

    @Override
    public void speak() {
        System.out.println("AndroidRobot says: Hello,
human!");
    }
}

// Separate components for composition example
static class Engine {
    void startEngine() {
        System.out.println("Engine started.");
    }
    String getStatus() {
        return "Engine running smoothly.";
```

```
    }
}

static class Transmission {
    void setGear(int gear) {
        System.out.println("Gear set to: " + gear);
    }
}

// A class that composes Engine and Transmission to exhibit
reusing multiple components
static class Car {
    private final Engine engine;
    private final Transmission transmission;

    Car(Engine engine, Transmission transmission) {
        this.engine = engine;
        this.transmission = transmission;
    }

    void start() {
        engine.startEngine();
    }

    void shiftGear(int gear) {
        transmission.setGear(gear);
    }

    void engineInfo() {
        System.out.println("Info: " + engine.getStatus());
    }
}

}
```

Output:

```
AndroidRobot is moving: walking humanoid steps.

AndroidRobot says: Hello, human!

Car status:

Engine started.

Gear set to: 3

Info: Engine running smoothly.
```

// Write a java program to implement the concept of interfaces and the final modifier

```
public class InterfaceFinalDemo {

public static void main(String[] args) {
    // Create a concrete vehicle that implements multiple
interfaces
    Vehicle vehicle = new Car("Sedan", 120);
    vehicle.start();
    vehicle.stop();
    System.out.println("Vehicle speed: " +
vehicle.getSpeed());

    // Show interface methods usage via a separate drone
    Flyable drone = new Drone("Quadcopter");
    drone.takeOff();
    drone.land();

    // Demonstrate final keyword usages
    System.out.println("Brand: " + CarBrand.BRAND);
    CarBrand brand = CarBrand.BRAND;
    // brand = CarBrand.OTHER_BRAND; // error: cannot assign
to final variable

    // A final class cannot be extended
    // class SportsCar extends Car { } // would cause error if
Car is final
}

// 1) Interfaces: define capabilities
interface Vehicle {
    void start();
    void stop();
    int getSpeed();
}

interface Flyable {
    void takeOff();
    void land();
}

// 2) A final class (cannot be subclassed)
final static class Car implements Vehicle {
    private final String model;
    private int speed; // mutable state
    private final String type;

    Car(String model, int maxSpeed) {
        this.model = model;
```

```
        this.speed = 0;
        this.type = "Passenger";
        // maxSpeed is not stored here; shown for
demonstration
    }

    // Overloaded constructor to set speed directly
    Car(String model, int speed, String type) {
        this.model = model;
        this.speed = speed;
        this.type = type;
    }

    @Override
    public void start() {
        speed = 10;
        System.out.println("Car " + model + " started. Speed="
+ speed);
    }

    @Override
    public void stop() {
        speed = 0;
        System.out.println("Car " + model + " stopped. Speed="
+ speed);
    }

    @Override
    public int getSpeed() {
        return speed;
    }

    public void accelerate(int delta) {
        speed += delta;
        System.out.println("Car " + model + " accelerated.
Speed=" + speed);
    }

    // A final method (cannot be overridden in subclasses)
    public final void honk() {
        System.out.println("Car " + model + ": HONK!");
    }

    // Getter for model
    public String getModel() {
        return model;
    }
}

// 3) Another class implementing Vehicle to show multiple
implementations
```

```
static class Bike implements Vehicle {
    private int speed = 0;

    @Override
    public void start() {
        speed = 8;
        System.out.println("Bike started. Speed=" + speed);
    }

    @Override
    public void stop() {
        speed = 0;
        System.out.println("Bike stopped. Speed=" + speed);
    }

    @Override
    public int getSpeed() {
        return speed;
    }
}

// 4) A separate class implementing Flyable
static class Drone implements Flyable {
    private final String id;

    Drone(String id) {
        this.id = id;
    }

    @Override
    public void takeOff() {
        System.out.println("Drone " + id + " taking off.");
    }

    @Override
    public void land() {
        System.out.println("Drone " + id + " landing.");
    }
}

// 5) Final class used in demonstration (string constant)
final static class CarBrand {
    static final CarBrand BRAND = new CarBrand("Swift
Motors");
    private final String name;

    private CarBrand(String name) {
        this.name = name;
    }

    @Override
```

```
    public String toString() {
        return "Brand: " + name;
    }
}

}
```

Output:

```
Car Sedan started. Speed=10

Car Sedan stopped. Speed=0

Vehicle speed: 0

Drone Quadricopter taking off.

Drone Quadricopter landing.

Brand: Swift Motors
```

// Write a Java program for;

// a) Command line arguments demonstration

// b) String class demonstrations (basic operations)

```
public class Demo {
    public static void main(String[] args) {
        System.out.println("=== Command Line Arguments ===");
        if (args.length == 0) {
            System.out.println("No command line arguments
provided.");
        } else {
            for (int i = 0; i < args.length; i++) {
                System.out.println("arg[" + i + "] = " +
args[i]);
            }
        }

        System.out.println("\n=== String Class Demonstrations
===");

        // 1) String creation and equals
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = new String("Hello");
        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
        System.out.println("s3: " + s3);
        System.out.println("s1 == s2: " + (s1 == s2)); // true
due to string interning
        System.out.println("s1 == s3: " + (s1 == s3)); //
false, different objects
```

```
        System.out.println("s1.equals(s3): " + s1.equals(s3));
// true, same content


        // 2) String immutability and concatenation

        String concat = s1 + " World";

        System.out.println("Concatenated: " + concat);


        // 3) String methods: length, charAt, substring,
indexOf, toUpperCase

        System.out.println("length of s1: " + s1.length());

        System.out.println("char at 1 in s1: " +
s1.charAt(1));

        System.out.println("substring(0,5) of s1: " +
s1.substring(0, 5));

        System.out.println("indexOf 'l' in s1: " +
s1.indexOf('l'));

        System.out.println("toUpperCase: " +
s1.toUpperCase());


        // 4) String formatting (printf-like)

        int a = 5, b = 7;

        System.out.printf("Formatted: a=%d, b=%d, a+b=%d%n",
a, b, a + b);


        // 5) StringBuilder for efficient mutations

        StringBuilder sb = new StringBuilder();

        sb.append("Mutable ")

          .append("String ")

          .append("Builder");

        System.out.println("StringBuilder content: " +
sb.toString());

    }

}
```

Output:

```
=== Command Line Arguments ===
No command line arguments provided.

=== String Class Demonstrations ===
s1: Hello
s2: Hello
s3: Hello
s1 == s2: true
s1 == s3: false
s1.equals(s3): true
Concatenated: Hello World
length of s1: 5
char at 1 in s1: e
substring(0,5) of s1: Hello
indexOf 'l' in s1: 2
toUpperCase: HELLO
Formatted: a=5, b=7, a+b=12
StringBuilder content: Mutable String Builder
```

//Write a java program to create a package and use it in another program

1.File: com/example/utility/Greeter.java

```
// File: src/com/example/utility/Greeter.java
package com.example.utility;
public class Greeter {
    private final String name;
    public Greeter(String name) {
        this.name = name;
    }
    public String greet() {
        return "Hello, " + name + "!";
    }
}
```

2.File: src/MainApp.java

```
// File: src/MainApp.java
import com.example.utility.Greeter;

public class MainApp {
    public static void main(String[] args) {
        Greeter greeter = new Greeter("Shreyash");
        System.out.println(greeter.greet());
    }
}
```

Output:

```
Hello, Shreyash!
```

// Write a simple Java program to demonstrate use of exception and user defined exception

```
public class ExceptionDemo {

public static void main(String[] args) {

    System.out.println("=== Demonstrating built-in exception (ArrayIndexOutOfBoundsException) ===");

    try {

        demonstrateArrayIndex();

    } catch (ArrayIndexOutOfBoundsException ex) {

        System.out.println("Caught built-in exception: " + ex);

    } finally {

        System.out.println("Finished built-in exception demonstration.\n");

    }

    System.out.println("=== Demonstrating user-defined exception (InvalidAgeException) ===");

    try {

        validateAge(25); // valid

        validateAge(-5); // invalid

    } catch (InvalidAgeException ex) {

        System.out.println("Caught user-defined exception: " + ex.getMessage());

    } finally {

        System.out.println("Finished user-defined exception demonstration.");

    }

}

// Method to intentionally trigger a built-in exception

static void demonstrateArrayIndex() {

    int[] arr = {1, 2, 3};

   // Accessing an index out of bounds to trigger the exception
```

```
    int value = arr[5];
    System.out.println("Value: " + value); // never reached
}
// Method that uses a user-defined exception
static void validateAge(int age) throws InvalidAgeException {
    if (age < 0 || age > 150) {
        throw new InvalidAgeException("Invalid age: " + age +
". Age must be between 0 and 150.");
    }
    System.out.println("Age " + age + " is valid.");
}
// 2) User-defined exception class (static nested class for a
single-file demo)
static class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}
}
```

Output:

```
=== Demonstrating built-in exception
(ArrayIndexOutOfBoundsException) ===

Caught built-in exception:
java.lang.ArrayIndexOutOfBoundsException: 5

Finished built-in exception demonstration.

=== Demonstrating user-defined exception (InvalidAgeException)
===

Age 25 is valid.

Caught user-defined exception: Invalid age: -5.

Age must be between 0 and 150.

Finished user-defined exception demonstration.
```

// **Write a java program to implement concept of multithreading**

```
public class ThreadSimpleDemo {
    public static void main(String[] args) {
        // Thread 1: extends Thread
        Thread t1 = new Thread() {
            public void run() {
                for (int i = 1; i <= 5; i++) {
                    System.out.println("Thread 1 - count " +
i);
                    sleepQuietly(100);
                }
            }
        };

        // Thread 2: implements Runnable
        Runnable r = () -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Thread 2 - count " + i);
                sleepQuietly(150);
            }
        };
        Thread t2 = new Thread(r);

        // Start threads
        t1.start();
        t2.start();

        // Main thread also prints, showing interleaving
        for (int i = 1; i <= 5; i++) {
            System.out.println("Main thread - count " + i);
```

```
            sleepQuietly(120);
        }

        // Wait for threads to finish (optional for this
simple demo)
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        System.out.println("Done.");
    }

    // helper to sleep without throwing checked exception in
main flow
    private static void sleepQuietly(long millis) {
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

Output:

```
Thread 1 - count 1
Thread 2 - count 1
Main thread - count 1
Thread 1 - count 2
Thread 2 - count 2
Main thread - count 2
Thread 1 - count 3
Thread 2 - count 3
Main thread - count 3
Thread 1 - count 4
Thread 2 - count 4
Main thread - count 4
Thread 1 - count 5
Thread 2 - count 5
Main thread - count 5
Done.
```