1. **1. Write a program to:**
   * **Read an int value from user input.**
   * **Assign it to a double (implicit widening) and print both.**
   * **Read a double, explicitly cast it to int, then to short, and print results—demonstrate truncation or overflow.**

package day6\_Assign;

import java.util.Scanner;

public class Q1 {

public static void main(String[]a){

Scanner s=new Scanner(System.***in***);

System.***out***.print("Enter int val:");

int i=s.nextInt();

double d=i;

System.***out***.println("int:"+i);

System.***out***.println("double:"+d);

System.***out***.print("Enter double val:");

double dd=s.nextDouble();

int ci=(int)dd;

short cs=(short)ci;

System.***out***.println("double:"+dd);

System.***out***.println("Int:"+ci);

System.***out***.println("Short:"+cs);

s.close();

}

}

**2. Convert an int to String using String.valueOf(...), then back with Integer.parseInt(...). Handle NumberFormatException**

package day6\_Assign;

public class Q2 {

public static void main(String[] args){

int orgVal=123;

String strVal=String.*valueOf*(orgVal);

System.***out***.println("String:"+strVal);

try{

int parsedVal=Integer.*parseInt*(strVal);

System.***out***.println("Parsed int:"+parsedVal);

}catch(NumberFormatException e){

System.***out***.println("Error:"+e.getMessage());

}

String invalidStr="abc";

try{

int parsedVal=Integer.*parseInt*(invalidStr);

System.***out***.println("Parsed int:"+parsedVal);

}catch(NumberFormatException e){

System.***out***.println("Error:"+e.getMessage());

}

}

}

3. Compound Assignment Behaviour

1. Initialize int x = 5;.
2. Write two operations:

x = x + 4.5; // Does this compile? Why or why not?

x += 4.5; // What happens here?

1. Print results and explain behavior in comments (implicit narrowing, compile error vs. successful assignment).

package day6\_Assign;

public class Q3{

public static void main(String[] args){

int x = 5;

// x = x + 4.5; // This will not compile because the result of x + 4.5 is a double

// System.out.println(x); // Uncommenting the above line will result in a compile error

x += 4.5; // This is equivalent to x = (int) (x + 4.5)

System.out.println(x); // Prints 9

}

}

**4. Object Casting with Inheritance**

1. **Define an Animal class with a method makeSound().**
2. **Define subclass Dog:**
   * **Override makeSound() (e.g. "Woof!").**
   * **Add method fetch().**
3. **In main:**

Dog d = new Dog();

Animal a = d; // upcasting

a.makeSound();

package day6\_Assign;

class Animal{

public void makeSound(){

System.out.println("The animal makes a sound.");

}

}

class Dog extends Animal{

public void makeSound(){

System.out.println("Woof!");

}

public void fetch(){

System.out.println("The dog fetches the ball.");

}

}

public class Main{

public static void main(String[]a){

Dog d=new Dog();

d.makeSound();

d.fetch();

Animal a=d;

a.makeSound();

if(a instanceof Dog){

Dog dog=(Dog)a;

dog.fetch();

}

}

}

5**. Mini‑Project – Temperature Converter**

1. **Prompt user for a temperature in Celsius (double).**
2. **Convert it to Fahrenheit:**

**double fahrenheit = celsius \* 9/5 + 32;**

1. **Then cast that fahrenheit to int for display.**
2. **Print both the precise (double) and truncated (int) values, and comment on precision loss.**

package day6\_Assign;

import java.util.Scanner;

public class Q5{

public static void main(String[]args){

Scanner scanner=new Scanner(System.in);

System.out.print("Enter Celsius:");

double celsius=scanner.nextDouble();

double fahrenheit=celsius\*9/5+32;

int fahrenheitInt=(int)fahrenheit;

System.out.println("Fahrenheit(double):"+fahrenheit);

System.out.println("Fahrenheit(int):"+fahrenheitInt);

scanner.close();

}}

6**. Enum**

**1: Days of the Week**

**Define an enum DaysOfWeek with seven constants. Then in main(), prompt the user to input a day name and**:

* Print its position via ordinal().

Confirm if it's a weekend day using a switch or if-statement.

package day6\_Assign;

import java.util.Scanner;

enum DaysOfWeek{

MONDAY,

TUESDAY,

WEDNESDAY,

THURSDAY,

FRIDAY,

SATURDAY,

SUNDAY;

}

public class Q1{

public static void main(String[]a){

Scanner s=new Scanner(System.in);

System.out.print("Enter day:");

String d=s.next().toUpperCase();

try{

DaysOfWeek day=DaysOfWeek.valueOf(d);

System.out.println(day+" is day "+day.ordinal());

if(day==DaysOfWeek.SATURDAY||day==DaysOfWeek.SUNDAY)

System.out.println(day+" is weekend");

else

System.out.println(day+" is weekday");

}catch(IllegalArgumentException e){

System.out.println("Invalid day");

}

s.close();

}

}

2. Compass Directions

**Create an enum Direction with the values NORTH, SOUTH, EAST, WEST. Write code to:**

* **Read a Direction from a string using valueOf().**

**Use switch or if to print movement (e.g. “Move north”).  
Test invalid inputs with proper error handling**

package day6\_Assign;

import java.util.Scanner;

import java.util.Scanner;

enum *Direction*{

*NORTH*,

*SOUTH*,

*EAST*,

*WEST*;

}

public class Q2{

public static void main(String[]a){

Scanner s=new Scanner(System.*in*);

System.*out*.print("Enter direction:");

String d=s.next().toUpperCase();

try{

*Direction* dir=*Direction*.*valueOf*(d);

switch(dir){

case *NORTH*:

System.*out*.println("Move north");

break;

case *SOUTH*:

System.*out*.println("Move south");

break;

case *EAST*:

System.*out*.println("Move east");

break;

case *WEST*:

System.*out*.println("Move west");

break;

}

}catch(IllegalArgumentException e){

System.*out*.println("Invalid direction,Enter valid one");

}

s.close();

}

}

**Shape Area Calculator**

**Define enum Shape (CIRCLE, SQUARE, RECTANGLE, TRIANGLE) where each constant:**

* **Overrides a method double area(double... params) to compute its area.**
* **E.g., CIRCLE expects radius, TRIANGLE expects base and height.  
  Loop over all constants with sample inputs and print results.**

3. Shape Area Calculator

package day6\_Assign;

enum *Shape* {

*CIRCLE* {

public double area(double... p) {

return Math.*PI* \* p[0] \* p[0];

}

},

*SQUARE* {

public double area(double... p) {

return p[0] \* p[0];

}

},

*RECTANGLE* {

public double area(double... p) {

return p[0] \* p[1];

}

},

*TRIANGLE* {

public double area(double... p) {

return 0.5 \* p[0] \* p[1];

}

};

public abstract double area(double... p);

}

public class Q8 {

public static void main(String[] a) {

double r = 5, s = 4, l = 6, b = 3;

for (*Shape* sh : *Shape*.*values*()) {

double ar = 0;

switch (sh) {

case *CIRCLE*:

ar = sh.area(r);

break;

case *SQUARE*:

ar = sh.area(s);

break;

case *RECTANGLE*:

ar = sh.area(l, b);

break;

case *TRIANGLE*:

ar = sh.area(l, b);

break;

}

System.*out*.println(sh+ " area: " + ar);

}

}

}

**.Card Suit & Rank**

**Redesign a Card class using two enums: Suit (CLUBS, DIAMONDS, HEARTS, SPADES) and Rank (ACE…KING).  
Then implement a Deck class to:**

* **Create all 52 cards.**
* **Shuffle and print the order**.

4. Card Suit & Rank

package day6\_Assign;

import java.util.Random;

enum *Suit* {

*CLUBS*,

*DIAMONDS*,

*HEARTS*,

*SPADES*

}

enum *Rank* {

*ACE*, *TWO*, *THREE*, *FOUR*, *FIVE*, *SIX*, *SEVEN*, *EIGHT*,

*NINE*, *TEN*, *JACK*, *QUEEN*, *KING*

}

class Card {

private *Suit* suit;

private *Rank* rank;

public Card(*Suit* suit, *Rank* rank) {

this.suit = suit;

this.rank = rank;

}

public String toString() {

return rank + " of " + suit;

}

}

class Deck {

private Card[] cards;

public Deck() {

cards = new Card[52];

int index = 0;

for (*Suit* s : *Suit*.*values*()) {

for (*Rank* r : *Rank*.*values*()) {

cards[index++] = new Card(s, r);

}

}

}

public void shuffle() {

Random r = new Random();

for (int i = cards.length - 1; i > 0; i--) {

int j = r.nextInt(i + 1);

Card temp = cards[i];

cards[i] = cards[j];

cards[j] = temp;

}

}

public void print() {

for (Card c : cards) {

System.*out*.println(c);

}

}

}

public class Q4 {

public static void main(String[] args) {

Deck d = new Deck();

d.shuffle();

d.print();

}

}

**5.**

**: Priority Levels with Extra Data**

**Implement enum PriorityLevel with constants (LOW, MEDIUM, HIGH, CRITICAL), each having:**

* **A numeric severity code.**
* **A boolean isUrgent() if severity ≥ some threshold.  
  Print descriptions and check urgency.**

**Priority Levels with Extra Data**

package day6\_Assign;

enum *PriorityLevel*{

*LOW*(1,false),

*MED*(2,false),

*HIGH*(3,true),

*CRIT*(4,true);

private int sev;

private boolean urg;

PriorityLevel(int s,boolean u){

sev=s;

urg=u;

}

public boolean isUrgent(){

return urg;

}

public int getSev(){

return sev;

}

}

public class Q5{

public static void main(String[]a){

for(*PriorityLevel* p:*PriorityLevel*.*values*()){

System.*out*.print(p+" sev:"+p.getSev());

if(p.isUrgent())

{

System.*out*.println(" urgent");

}

else

{

System.*out*.println(" not urgent");

}

}

}

}

6. **Traffic Light State Machine**

**Implement enum TrafficLight implementing interface State, with constants RED, GREEN, YELLOW.  
Each must override State next() to transition in the cycle.  
Simulate and print six transitions starting from RED.**

package day6\_Assign;

interface State {

State next();

}

enum TrafficLight implements State {

RED {

public State next() {

return GREEN;

}

},

GREEN {

public State next() {

return YELLOW;

}

},

YELLOW {

public State next() {

return RED;

}

};

}

public class Q6 {

public static void main(String[] args) {

State light = TrafficLight.RED;

for (int i = 0; i < 6; i++) {

System.out.println(light);

light = light.next();

}

}

}

7. **Difficulty Level & Game Setup**

**Define enum Difficulty with EASY, MEDIUM, HARD.  
Write a Game class that takes a Difficulty and prints logic like:**

**EASY → 3000 bullets, MEDIUM → 2000, HARD → 1000.  
Use a switch(diff) inside constructor or method.**

**package day6\_Assign;**

enum *Difficulty* {

*EASY*,

*MEDIUM*,

*HARD*;

}

class Game {

private int bullets;

public Game(*Difficulty* diff) {

switch (diff) {

case *EASY*:

bullets = 3000;

break;

case *MEDIUM*:

bullets = 2000;

break;

case *HARD*:

bullets = 1000;

break;

}

System.*out*.println(diff + " mode: bullets=" + bullets);

}

}

public class Q7{

public static void main(String[] args) {

new Game(*Difficulty*.*EASY*);

new Game(*Difficulty*.*MEDIUM*);

new Game(*Difficulty*.*HARD*);

}

}

**8.Calculator Operations Enum**

**Create enum Operation (PLUS, MINUS, TIMES, DIVIDE) with an eval(double a, double b) method.  
Implement two versions:**

* **One using a switch(this) inside eval.**

**Another using constant-specific method overrides for eval.  
Compare both designs**

package day6\_Assign;

enum Operation {

PLUS {

public double eval(double a, double b) {

return a + b;

}

},

MINUS {

public double eval(double a, double b) {

return a - b;

}

},

TIMES {

public double eval(double a, double b) {

return a \* b;

}

},

DIVIDE {

public double eval(double a, double b) {

if (b == 0) {

throw new ArithmeticException("Cannot divide by zero");

}

return a / b;

}

};

public abstract double eval(double a, double b);

public static void main(String[] args) {

System.out.println(Operation.PLUS.eval(2, 3));

System.out.println(Operation.MINUS.eval(5, 2));

System.out.println(Operation.TIMES.eval(4, 5));

System.out.println(Operation.DIVIDE.eval(10, 2));

}

}

10. **Knowledge Level from Score Range**

**Define enum KnowledgeLevel with constants BEGINNER, ADVANCED, PROFESSIONAL, MASTER.  
Use a static method fromScore(int score) to return the appropriate enum:**

**0–3 → BEGINNER, 4–6 → ADVANCED, 7–9 → PROFESSIONAL, 10 → MASTER.  
Then print the level and test boundary conditions**.

enum KnowledgeLevel {

BEGINNER(0, 3),

ADVANCED(4, 6),

PROFESSIONAL(7, 9),

MASTER(10, 10);

private int minScore;

private int maxScore;

KnowledgeLevel(int minScore, int maxScore) {

this.minScore = minScore;

this.maxScore = maxScore;

}

public static KnowledgeLevel fromScore(int score) {

for (KnowledgeLevel level : values()) {

if (score >= level.minScore && score <= level.maxScore) {

return level;

}

}

throw new IllegalArgumentException("Invalid score");

}

public static void main(String[] args) {

System.out.println(KnowledgeLevel.fromScore(2));

System.out.println(KnowledgeLevel.fromScore(5));

System.out.println(KnowledgeLevel.fromScore(8));

System.out.println(KnowledgeLevel.fromScore(10));

}

}

Exception handling

**1.Division & Array Access**

**Division & Array Access**

**Write a Java class ExceptionDemo with a main method that:**

1. **Attempts to divide an integer by zero and access an array out of bounds.**
2. **Wrap each risky operation in its own try‑catch:**
   * **Catch only the specific exception types: ArithmeticException and ArrayIndexOutOfBoundsException.**
   * **In each catch, print a user-friendly message.**
3. **Add a finally block after each try‑catch that prints "Operation completed.".**

**Example structure:**

**try {**

**// division or array access**

**} catch (ArithmeticException e) {**

**System.out.println("Division by zero is not allowed!");**

**} finally {**

**System.out.println("Operation completed.");**

**}**

package day6\_Assign;

public class Exc1 {

public static void main(String[] args) {

try {

int a = 10;

int b = 0;

int res = a / b;

System.*out*.println("res:" + res);

} catch (ArithmeticException e) {

System.*out*.println("division by zero is not allowed!");

} finally {

System.*out*.println("operation completed.");

}

try {

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

int value = nums[5];

System.*out*.println("Val:" + value);

} catch (ArrayIndexOutOfBoundsException e) {

System.*out*.println("array index is out of bounds!");

} finally {

System.*out*.println("operation completed.");

}

}

}

2. Throw and Handle Custom Exception

package day6\_Assign;

public class OddNumExc extends Exception {

public OddNumExc(String msg) {

super(msg);

}

}

package day6\_Assign;

public class ChkOddNum {

public static void chkOdd(int n) throws OddNumExc {

if (n % 2 != 0) {

throw new OddNumExc("Odd num:" + n);

}

}

public static void main(String[] args) {

int[] nums = {2, 5, 8, 11, 14};

for (int n : nums) {

try {

*chkOdd*(n);

System.*out*.println(n + " is even");

} catch (OddNumExc e) {

System.*out*.println(e.getMessage());

}

}

}

}

File Handling with Multiple Catches

package day6\_Assign;

import java.io.\*;

public class FileReadDemo {

public static void readLn(String filename) throws FileNotFoundException, IOException {

BufferedReader br = new BufferedReader(new FileReader(filename));

String firstLine = br.readLine();

System.out.println("First line:" + firstLine);

br.close();

}

public static void main(String[] args) {

String filename = "test.txt";

try {

readLn(filename);

}

catch (FileNotFoundException e) {

System.out.println("File not found:" + filename);

}

catch (IOException e) {

System.out.println("Error reading file:" + e.getMessage());

}

finally {

System.out.println("done");

}

}

}

4**.Multi‑Exception in One Try Block**

**Write a class MultiExceptionDemo:**

* **In a single try block, perform:**
  + **Opening a file**
  + **Parsing its first line as integer**
  + **Dividing 100 by that integer**
* **Use multiple catch blocks in this order:**
  + **FileNotFoundException**
  + **IOException**
  + **NumberFormatException**
  + **ArithmeticException**
* **In each catch, print a tailored message:**
  + **File not found**
  + **Problem reading file**
  + **Invalid number format**
  + **Division by zero**
* **Finally, print "Execution completed".**

package day6\_Assign;

import java.io.\*;

public class MultiExcDemo {

public static void main(String[] args) {

String filename = "test.txt";

try {

BufferedReader br = new BufferedReader(new FileReader(filename));

String l = br.readLine();

br.close();

int n = Integer.parseInt(l);

int res = 100 / n;

System.out.println("Res:" + res);

} catch (FileNotFoundException e) {

System.out.println("File not found");

}

catch (IOException e) {

System.out.println("Problem reading file");

}

catch (NumberFormatException e) {

System.out.println("Invalid number format");

}

catch (ArithmeticException e) {

System.out.println("Div by zero");

}

finally {

System.out.println("Execution completed");

}

}

}