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 Assignment_Day6;

import java.util.Scanner;

public class Problem1 {

public static void main(String[] args) {
// TODO Auto-generated method stub
Scanner sc=new Scanner(System.in);
int num=sc.nextInt();
double d=num;
double d1=sc.nextDouble();
int num1=(int)d1;
System.out.println("Int value is "+num);
System.out.println("Double value is "+d);
System.out.println("Double value is "+d1);
System.out.println("Int value is "+num1);
}

}
```

```
20
20.2
Int value is 20
Double value is 20.0
Double value is  20.2
Int value is 20
```

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

```
package Assignment_Day6;

public class Problem2 {

public static void main(String[] args) {
// TODO Auto-generated method stub
int num=1234;
String s=String.valueOf(num);
System.out.println("String: "+s);
try {
int cint=Integer.parseInt(s);
System.out.println("Back to int: "+cint);
}
catch(Exception e) {
System.out.println(e);
}
}

}
```

```
String: 1234
Back to int: 1234
```

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?

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

x = x + 4.5; does not compile because x is int and 4.5 is double

Where java performs implicit conversion but after conversion we are again assigning it to int beacuse we cannot assign double to int.

x+=4.5;which works as like x=(int)(x+4.5); so it does not raises any error

Where the output is 9.

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();

```
class Animal {
```

```
void makeSound() {
System.out.println("animal sound");
}
}

class Dog extends Animal {
void makeSound() {
System.out.println("Woof!");
}
void fetch() {
System.out.println("Fetching the ball...");
}
}

public class Overriding_Practice {
public static void main(String[] args) {
Dog d = new Dog();
Animal a = d;
a.makeSound();
}
}
```

```
Woof!
```

```
package Assignment_Day6;

import java.util.Scanner;

public class Problem5 {

public static void main(String[] args) {
// TODO Auto-generated method stub
 Scanner sc = new Scanner(System.in);
 System.out.print("temperature in Celsius: ");
 double celsius = sc.nextDouble();
 double fahrenheit = celsius * 9.0/5 + 32;
 int fahrenheitTruncated = (int)fahrenheit;
 System.out.printf(" Fahrenheit in double : %.2fF\n", fahrenheit);
 System.out.println("Fahrenheit in int: " + fahrenheitTruncated + "F");



}

}
```

```
temperature in Celsius: 23.2
 Fahrenheit in double : 73.76F
Fahrenheit in int: 73F
```

We lost the floating point values after conversion of double to int.

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 Enum;

import java.util.Scanner;

import Enum.Enum3.Day;

public class Enum4 {
enum Day{Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday}
public static void main(String[] args) throws IllegalArgumentException {
Scanner sc=new Scanner(System.in);
String s=sc.next();
Day d=Day.valueOf(s);

System.out.println((d==Day.Saturday || d==Day.Sunday)? "WEEKEND":"WEEKDAY");
switch(d)
{
case Monday:System.out.println("Monday");
break;
case Tuesday:System.out.println("Tuesday");
break;
```

```
case Wednesday:System.out.println("Wednesday");
break;
case Thursday:System.out.println("Thursday");
break;
case Friday:System.out.println("Friday");
break;
case Saturday:System.out.println("Saturday");
break;
case Sunday:System.out.println("Sunday");
break;
}

}

}
```

```
Saturday
WEEKEND
Saturday
```

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 Assignment_Day6;
```

```java
import java.util.Scanner;

public class Problem6 {
enum Direction{NORTH,SOUTH,EAST,WEST}
public static void main(String[] args) throws IllegalArgumentException {
// TODO Auto-generated method stub
Scanner sc = new Scanner(System.in);
System.out.print("Enter a direction :");
String input = sc.nextLine();
Direction direction = Direction.valueOf(input);
switch(direction) {
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;
}

}

}
```

```
Enter a direction :NORTH
Move north
```

3: 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.

```
package Assignment_Day6;

public class Problem7 {
enum Shape {
CIRCLE {
double area(double... params) {
if (params.length != 1) throw new IllegalArgumentException("Circle requires 1
parameter (radius)");
return Math.PI * params[0] * params[0];
}
},
SQUARE {
@Override
double area(double... params) {
if (params.length != 1) throw new IllegalArgumentException("Square requires 1
parameter (side)");
return params[0] * params[0];
}
},
RECTANGLE {
@Override
double area(double... params) {
if (params.length != 2) throw new IllegalArgumentException("Rectangle requires 2
parameters (length, width)");
return params[0] * params[1];
}
},
TRIANGLE {
@Override
double area(double... params) {
if (params.length != 2) throw new IllegalArgumentException("Triangle requires 2
parameters (base, height)");
return 0.5 * params[0] * params[1];
}
};

abstract double area(double... params);
```

```
}

public static void main(String[] args) {
double[] circleParams = {5.0};
double[] squareParams = {4.0};
double[] rectangleParams = {6.0, 3.0};
double[] triangleParams = {4.0, 7.0};

for (Shape shape : Shape.values()) {
try {
double area = 0;
switch (shape) {
case CIRCLE:
area = shape.area(circleParams[0]);
break;
case SQUARE:
area = shape.area(squareParams[0]);
break;
case RECTANGLE:
area = shape.area(rectangleParams[0], rectangleParams[1]);
break;
case TRIANGLE:
area = shape.area(triangleParams[0], triangleParams[1]);
break;
}
System.out.printf(" %s area =%.2f%n", shape, area);
} catch (IllegalArgumentException e) {
System.out.println(e);
}
}
}
}
```

```
CIRCLE area =78.54
SQUARE area =16.00
RECTANGLE area =18.00
TRIANGLE area =14.00
```

4.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.

```
package Assignment_Day6;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Problem8 {

enum Suit {
CLUBS, DIAMONDS, HEARTS, SPADES
}

enum Rank {
ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT,
NINE, TEN, JACK, QUEEN, KING
}



static class Card {
private final Suit suit;
private final Rank rank;

public Card(Suit suit, Rank rank) {
this.suit = suit;
this.rank = rank;
}

@Override
public String toString() {
return rank + " of " + suit;
}
}
```

```
static class Deck {
private final List<Card> cards;

public Deck() {
cards = new ArrayList<>();
for (Suit suit : Suit.values()) {
for (Rank rank : Rank.values()) {
cards.add(new Card(suit, rank));
}
}
}

public void shuffle() {
Collections.shuffle(cards);
}

public void printDeck() {
for (Card card : cards) {
System.out.println(card);
}
}
}

public static void main(String[] args) {
Deck deck = new Deck();
System.out.println("=== Original Deck ===");
deck.printDeck();
System.out.println("\n=== Shuffled Deck ===");
deck.shuffle();
deck.printDeck();
}
}
```

```
=== Original Deck ===
ACE of CLUBS
TWO of CLUBS
THREE of CLUBS
FOUR of CLUBS
FIVE of CLUBS
SIX of CLUBS
SEVEN of CLUBS
EIGHT of CLUBS
NINE of CLUBS
TEN of CLUBS
JACK of CLUBS
QUEEN of CLUBS
KING of CLUBS
ACE of DIAMONDS
TWO of DIAMONDS
THREE of DIAMONDS
FOUR of DIAMONDS
FIVE of DIAMONDS
SIX of DIAMONDS
SEVEN of DIAMONDS
EIGHT of DIAMONDS
NINE of DIAMONDS
TEN of DIAMONDS
JACK of DIAMONDS
QUEEN of DIAMONDS
KING of DIAMONDS
ACE of HEARTS
TWO of HEARTS
THREE of HEARTS
FOUR of HEARTS
FIVE of HEARTS
SIX of HEARTS
SEVEN of HEARTS
EIGHT of HEARTS
NINE of HEARTS
TEN of HEARTS
JACK of HEARTS
QUEEN of HEARTS
KING of HEARTS
```

```
EIGHT of SPADES
NINE of SPADES
TEN of SPADES
JACK of SPADES
QUEEN of SPADES
KING of SPADES

=== Shuffled Deck ===
THREE of DIAMONDS
FIVE of SPADES
THREE of HEARTS
TEN of HEARTS
NINE of DIAMONDS
ACE of HEARTS
JACK of CLUBS
FOUR of CLUBS
SEVEN of DIAMONDS
SEVEN of SPADES
THREE of CLUBS
SIX of SPADES
FOUR of HEARTS
FIVE of CLUBS
KING of CLUBS
FOUR of DIAMONDS
EIGHT of DIAMONDS
JACK of DIAMONDS
SEVEN of HEARTS
THREE of SPADES
ACE of SPADES
TWO of DIAMONDS
TWO of HEARTS
FIVE of DIAMONDS
SEVEN of CLUBS
FOUR of SPADES
NINE of HEARTS
SIX of DIAMONDS
NINE of SPADES
FIVE of HEARTS
QUEEN of HEARTS
```

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.

```
package Enum;

public class Enum2 {
enum Level{High,Medium,Low}
public static void main(String[] args) {
Level l=Level.Low;
switch(l)
{
case Low:System.out.println("Low Level");
break;
case Medium:System.out.println("Medium Level");
break;
case High:System.out.println("High Level");
break;
}


}

}
```

```
Low Level
```

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 Assignment_Day6;
```

```
interface State {
State next();
}

enum TrafficLight implements State {
RED {
@Override
public State next() {
System.out.println("RED -> GREEN");
return GREEN;
}
},
GREEN {
@Override
public State next() {
System.out.println("GREEN -> YELLOW");
return YELLOW;
}
},
YELLOW {
@Override
public State next() {
System.out.println("YELLOW -> RED");
return RED;
}
};
}

public class Problem9 {
public static void main(String[] args) {
State currentLight = TrafficLight.RED;
System.out.println("Starting traffic light simulation from RED:");
for (int i = 0; i < 6; i++) {
currentLight = currentLight.next();
}
}
}
```

```
Starting traffic light simulation from RED:
RED -> GREEN
GREEN -> YELLOW
YELLOW -> RED
RED -> GREEN
GREEN -> YELLOW
YELLOW -> RED
```

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 Assignment_Day6;

enum Difficulty {
EASY,
MEDIUM,
HARD
}

class Game {
private int bulletCount;
private Difficulty difficulty;

public Game(Difficulty difficulty) {
this.difficulty = difficulty;
setupGame();
}

private void setupGame() {
switch (difficulty) {
case EASY:
bulletCount = 3000;
break;
case MEDIUM:
bulletCount = 2000;
break;
```

```
case HARD:
bulletCount = 1000;
break;
}
}

public void printGameSetup() {
System.out.println("Difficulty: " + difficulty);
System.out.println("Bullet count: " + bulletCount);
System.out.println("Enemy strength: " + getEnemyStrength());
}

private String getEnemyStrength() {
return switch (difficulty) {
case EASY -> "Weak";
case MEDIUM -> "Moderate";
case HARD -> "Strong";
};
}
}

public class Problem10 {
public static void main(String[] args) {
Game easyGame = new Game(Difficulty.EASY);
Game mediumGame = new Game(Difficulty.MEDIUM);
Game hardGame = new Game(Difficulty.HARD);
System.out.println("=== Easy Game ===");
easyGame.printGameSetup();
System.out.println("\n=== Medium Game ===");
mediumGame.printGameSetup();
System.out.println("\n=== Hard Game ===");
hardGame.printGameSetup();
}
}
```

```
=== Medium Game ===
Difficulty: MEDIUM
Bullet count: 2000
Enemy strength: Moderate

=== Hard Game ===
Difficulty: HARD
Bullet count: 1000
Enemy strength: Strong
```

Exception handling

1: 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 trycatch:
    - Catch only the specific exception types: ArithmeticException and ArrayIndexOutOfBoundsException.
    - In each catch, print a user-friendly message.
3. Add a finally block after each trycatch that prints "Operation completed.".

```
package Assignment_Day6;

public class Problem11 {
public static void main(String[] args) {

try {
int numerator = 10;
int denominator = 0;
int result = numerator / denominator;
System.out.println("Division result: " + result);
} catch (ArithmeticException e) {
System.out.println("Division by zero error");
} finally {
System.out.println("Operation completed.");
}

try {
int[] numbers = {1, 2, 3};
System.out.println("Array element: " + numbers[5]);
} catch (ArrayIndexOutOfBoundsException e) {
System.out.println("Array index is out of bounds");
```

```
} finally {
System.out.println("Operation completed.");
}
}
}
```

```
Division by zero error
Operation completed.
Array index is out of bounds
Operation completed.
```

2: Throw and Handle Custom Exception

Create a class OddChecker:

1. Implement a static method:

public static void checkOdd(int n) throws OddNumberException { /* ... */ }

2. If n is odd, throw a custom checked exception OddNumberException with message "Odd number: " + n.
3. In main:
    - Call checkOdd with different values (including odd and even).
    - Handle exceptions with trycatch, printing e.getMessage() when caught.

Define the exception like:

public class OddNumberException extends Exception {

public OddNumberException(String message) { super(message); }

}

```
package Assignment_Day6;


class OddNumberException extends Exception {
public OddNumberException(String message) {
super(message);
}
}

public class Problem12 {

public static void checkOdd(int n) throws OddNumberException {
if (n % 2 != 0) {
throw new OddNumberException("Odd number: " + n);
}
System.out.println(n + " is even (no exception thrown)");
}

public static void main(String[] args) {
int[] testNumbers = {2, 3, 4, 5, 6, 7};
for (int num : testNumbers) {
try {
checkOdd(num);
} catch (OddNumberException e) {
System.out.println("Caught exception: " + e.getMessage());
}
}
}
}
```

```
2 is even (no exception thrown)
Caught exception: Odd number: 3
4 is even (no exception thrown)
Caught exception: Odd number: 5
6 is even (no exception thrown)
Caught exception: Odd number: 7
```

MultiException 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 Assignment_Day6;
```

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Problem13 {
public static void main(String[] args) {
try {
File file = new File("numbers.txt");
BufferedReader reader = new BufferedReader(new FileReader(file));
String firstLine = reader.readLine();
int number = Integer.parseInt(firstLine);
int result = 100 / number;
System.out.println("100 divided by " + number + " is " + result);
reader.close();
} 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("Division by zero");
} finally {
System.out.println("Execution completed");
}
}
}
```

```
File not found
Execution completed
```