**Day-6 Assignment**

1. Write a program to:

* Read an int value from the 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

**Program:**

**import** java.util.Scanner;

**public** **class** TypeCastingDemo

{

**public** **static** **void** main(String[] args)

{

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

System.***out***.print("Enter an integer value: ");

**int** intValue = sc.nextInt();

**double** widenedValue = intValue;

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

System.***out***.println("Widened to double: " + widenedValue);

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

**double** doubleValue = sc.nextDouble();

**int** intFromDouble = (**int**) doubleValue;

**short** shortFromDouble = (**short**) intFromDouble;

System.***out***.println("Original double value: " + doubleValue);

System.***out***.println("After casting to int: " + intFromDouble);

System.***out***.println("After casting to short: " + shortFromDouble);

sc.close();

}

}

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

**Program:**

**public** **class** IntStringConversion

{

**public** **static** **void** main(String[] args)

{

**int** number = 123;

String strNumber = String.*valueOf*(number);

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

**try** {

**int** parsedNumber = Integer.*parseInt*(strNumber);

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

String invalidStr = "123abc";

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

System.***out***.println("This won't print: " + invalidParsed);

}

**catch** (NumberFormatException e)

{

System.***out***.println("Error: Invalid number format - " + e.getMessage());

}

}

}

1. Compound Assignment Behaviour

* Initialize int x = 5;
* Write two operations:

x = x + 4.5; // Does this compile? Why? Or Why Not?

x+ = 4.5; // What happens here?

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

**Program:**

**public** **class** CompoundAssignmentDemo

{

**public** **static** **void** main(String[] args)

{

**int** x = 5;

x = (**int**) (x + 4.5); // Explicit narrowing from double → int

System.***out***.println("After explicit cast (x = x + 4.5): " + x); // Output: 9 (truncated)

x = 5;

x += 4.5;

System.***out***.println("After compound assignment (x += 4.5): " + x); // Output: 9

}

}

1. Object Casting with Inheritance

* Define an Animal class with a method makeSound().
* Define sub-class Dog:
  + - * Override makeSound() (e.g., “Woof!”)
      * Add method fetch()
* In main:

Dog d = new Dog();

Animal a = d; // upcasting

a.makeSound();

**Program:**

**class** Animal

{

**void** makeSound()

{

System.***out***.println("Animal makes a sound");

}

}

**class** Dog **extends** Animal

{

@Override

**void** makeSound()

{

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

}

**void** fetch()

{

System.***out***.println("Dog is fetching the ball...");

}

}

**public** **class** CastingDemo

{

**public** **static** **void** main(String[] args)

{

Dog d = **new** Dog();

Animal a = d;

a.makeSound();

Dog d2 = (Dog) a;

d2.fetch();

}

}

1. Mini Project – Temperature Converter

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

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

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

**Program:**

**import** java.util.Scanner;

**public** **class** TemperatureConverter

{

**public** **static** **void** main(String[] args)

{

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

System.***out***.print("Enter temperature in Celsius: ");

**double** celsius = sc.nextDouble();

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

**int** fahrenheitInt = (**int**) fahrenheit;

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

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

}

}

**Enum:**

1. Days of the week

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

* Print its position via ordinal()
* Confirm if it’s a weekend using switch or if-statement

**Program:**

**import** java.util.Scanner;

**public** **class** EnumExample

{

**enum** DaysOfWeek

{

***MONDAY***, ***TUESDAY***, ***WEDNESDAY***, ***THURSDAY***, ***FRIDAY***, ***SATURDAY***, ***SUNDAY***

}

**public** **static** **void** main(String[] args)

{

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

System.***out***.print("Enter a day name (e.g., MONDAY): ");

String input = sc.next().toUpperCase();

**try**

{

DaysOfWeek day = DaysOfWeek.*valueOf*(input);

System.***out***.println("Position (ordinal): " + day.ordinal());

**switch** (day) {

**case** ***SATURDAY***:

**case** ***SUNDAY***:

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

**break**;

**default**:

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

}

}

**catch** (IllegalArgumentException e)

{

System.***out***.println("Invalid day entered. Please enter a valid day name.");

}

}

}

1. 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.

**Program:**

**import** java.util.Scanner;

**public** **class** CompassExample

{

**enum** Direction

{

***NORTH***, ***SOUTH***, ***EAST***, ***WEST***

}

**public** **static** **void** main(String[] args)

{

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

System.***out***.print("Enter a direction (NORTH, SOUTH, EAST, WEST): ");

String input = sc.next().toUpperCase();

**try** {

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

**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! Please enter NORTH, SOUTH, EAST, or WEST.");

}

}

}

1. 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.

**Program:**

**public** **class** ShapeAreaCalculator

{

// 1. Define enum with abstract method

**enum** Shape

{

***CIRCLE***

{

@Override

**double** area(**double**... params)

{

**if** (params.length < 1) **throw** **new** IllegalArgumentException("Circle needs radius");

**double** radius = params[0];

**return** Math.***PI*** \* radius \* radius;

}

},

***SQUARE***

{

@Override

**double** area(**double**... params)

{

**if** (params.length < 1) **throw** **new** IllegalArgumentException("Square needs side length");

**double** side = params[0];

**return** side \* side;

}

},

***RECTANGLE***

{

@Override

**double** area(**double**... params)

{

**if** (params.length < 2) **throw** **new** IllegalArgumentException("Rectangle needs length and width");

**double** length = params[0];

**double** width = params[1];

**return** length \* width;

}

},

***TRIANGLE***

{

@Override

**double** area(**double**... params)

{

**if** (params.length < 2) **throw** **new** IllegalArgumentException("Triangle needs base and height");

**double** base = params[0];

**double** height = params[1];

**return** 0.5 \* base \* height;

}

};

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

}

**public** **static** **void** main(String[] args)

{

**for** (Shape shape : Shape.*values*())

{

**double** result;

**switch** (shape)

{

**case** ***CIRCLE***:

result = shape.area(5); // radius

**break**;

**case** ***SQUARE***:

result = shape.area(4); // side

**break**;

**case** ***RECTANGLE***:

result = shape.area(4, 6); // length, width

**break**;

**case** ***TRIANGLE***:

result = shape.area(3, 8); // base, height

**break**;

**default**:

**throw** **new** IllegalStateException("Unexpected shape: " + shape);

}

System.***out***.printf("%s area = %.2f%n", shape, result);

}

}

}

1. Card Suite & 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.

**Program:**

**import** java.util.\*;

//Enum for suits

**enum** Suit

{

***CLUBS***, ***DIAMONDS***, ***HEARTS***, ***SPADES***

}

**enum** Rank

{

***ACE***, ***TWO***, ***THREE***, ***FOUR***, ***FIVE***, ***SIX***, ***SEVEN***, ***EIGHT***, ***NINE***, ***TEN***, ***JACK***, ***QUEEN***, ***KING***

}

//Card class using enums

**class** Card

{

**private** **final** Suit suit;

**private** **final** Rank rank;

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

{

**this**.suit = suit;

**this**.rank = rank;

}

**public** Suit getSuit()

{

**return** suit;

}

**public** Rank getRank()

{

**return** rank;

}

@Override

**public** String toString()

{

**return** rank + " of " + suit;

}

}

//Deck class

**class** Deck

{

**private** **final** List<Card> cards = **new** ArrayList<>();

**public** Deck()

{

// Create all 52 cards

**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);

}

}

}

//Main class

**public** **class** CardGame

{

**public** **static** **void** main(String[] args)

{

Deck deck = **new** Deck();

System.***out***.println("Before shuffling:");

deck.printDeck();

deck.shuffle();

System.***out***.println("\nAfter shuffling:");

deck.printDeck();

}

}

1. 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.

**Program:**

**enum** PriorityLevel

{

***LOW***(1),

***MEDIUM***(2),

***HIGH***(3),

***CRITICAL***(4);

**private** **final** **int** severityCode;

PriorityLevel(**int** severityCode)

{

**this**.severityCode = severityCode;

}

**public** **int** getSeverityCode()

{

**return** severityCode;

}

**public** **boolean** isUrgent()

{

**return** severityCode >= 3; // HIGH and CRITICAL are urgent

}

@Override

**public** String toString()

{

**return** name() + " (Severity " + severityCode + ")";

}

}

**public** **class** PriorityTest

{

**public** **static** **void** main(String[] args)

{

**for** (PriorityLevel level : PriorityLevel.*values*())

{

System.***out***.println(level +

" | Urgent? " + level.isUrgent());

}

}

}

1. 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.

**Program:**

**interface** State

{

State next();

}

**enum** TrafficLight **implements** State

{

***RED***

{

@Override

**public** State next()

{

**return** ***GREEN***;

}

},

***GREEN***

{

@Override

**public** State next()

{

**return** ***YELLOW***;

}

},

***YELLOW***

{

@Override

**public** State next()

{

**return** ***RED***;

}

};

}

**public** **class** TrafficLightSimulation

{

**public** **static** **void** main(String[] args)

{

State current = TrafficLight.***RED***; // Starting point

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

{

System.***out***.println("Current Light: " + current);

current = current.next(); // Move to the next state

}

}

}