1. **Answer**

import java.util.Scanner;

public class Typecast {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("enter an int value: ");

int intValue = scanner.nextInt();

double doubleValue = intValue;

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

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

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

double inputDouble = scanner.nextDouble();

int castedInt = (int) inputDouble;

short castedShort = (short) castedInt;

System.out.println("original value: " + inputDouble);

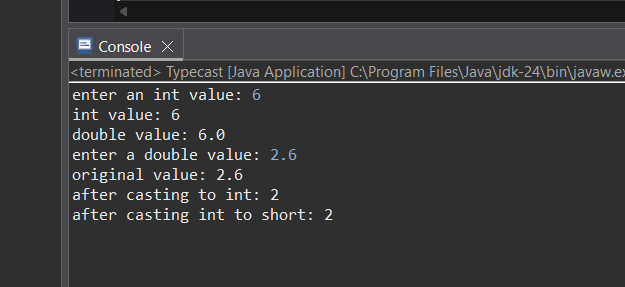
System.out.println("after casting to int: " + castedInt);

System.out.println("after casting int to short: " + castedShort);

}

}

**OUTPUT:**



1. **Answer**

public class Convers {

public static void main(String[] args) {

int originalInt = 12345;

String strValue = String.valueOf(originalInt);

System.out.println("string value: " + strValue);

try {

int parsedInt = Integer.parseInt(strValue);

System.out.println("parsed int value: " + parsedInt);

} catch (NumberFormatException e) {

System.out.println("string is not a valid integer");

e.printStackTrace();

}

String invalidStr = "12a34";

try {

int invalidParse = Integer.parseInt(invalidStr);

System.out.println("parsed int value: " + invalidParse);

} catch (NumberFormatException e) {

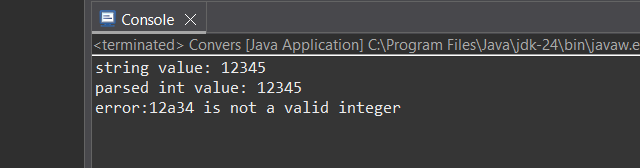
System.out.println("error:" + invalidStr + " is not a valid integer");

}

}

}

**OUTPUT:**



1. **Answer**

public static void main(String[] args) {

int x = 5;

x = x + 4.5; // This line doesnot compile because:

x += 4.5;

System.out.println("Result of x += 4.5: " + x);

Output: 9

// Explanation: 5 + 4.5 = 9.5, but implicit cast to int truncates decimal part, so result is 9.

}

}

**Explanation:**

x = x + 4.5;

The expression (x + 4.5) results in a double,

and Java does not allow implicit narrowing from double to int.

So assigning a double to int directly without casting causes a compile-time error.

Using compound assignment:

x += 4.5;

the above one compiles successfully because compound assignment operator perform an implicit cast to the type of the left-hand variable.

1. **Answer**

class Animal {

public void makeSound() {

System.***out***.println("animal sound");

}

}

class Dog extends Animal {

*@Override*

public void makeSound() {

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

}

public void fetch() {

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

}

}

public class Inherit {

public static void main(String[] args) {

Dog d = new Dog();

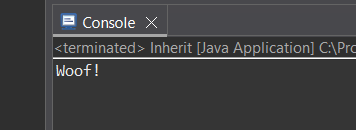
Animal a = d;

a.makeSound();

}

}

**OUTPUT:**



**Enum**

**1: Days of the Week**

import java.util.Scanner;

enum *DaysOfWeek* {

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

}

public class Days {

public static void main(String[] args) {

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

System.***out***.print("enter day of week: ");

String input = scanner.nextLine().trim().toUpperCase();

try {

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

System.***out***.println("position of " + day + " is: " + 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) {

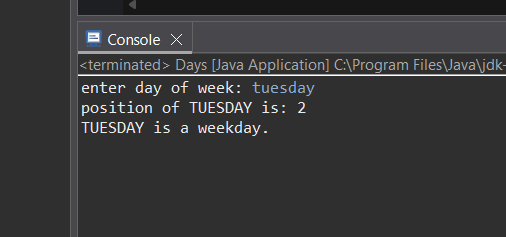
System.***out***.println("invalid day entered");

}

}

}

**OUTPUT:**



**2. Answer**

import java.util.Scanner;

enum *Direction* {

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

}

public class Directions {

public static void main(String[] args) {

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

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

String input = scanner.nextLine().trim().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 enteredn");

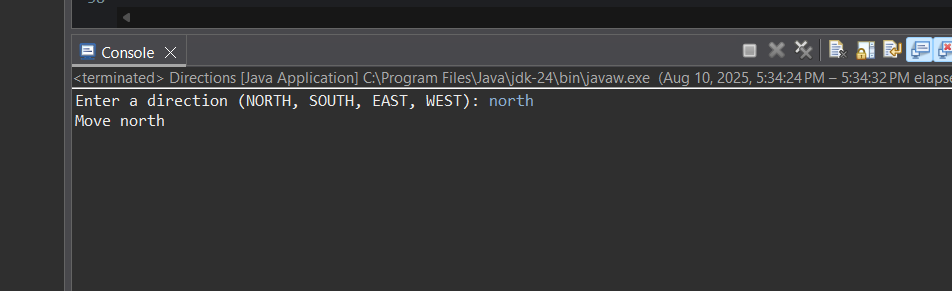
}

scanner.close();

}

}

**OUTPUT:**



**3. Answer**

enum *Shape* {

***CIRCLE*** {

*@Override*

double area(double... params) {

double radius = params[0];

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

}

},

***SQUARE*** {

*@Override*

double area(double... params) {

double side = params[0];

return side \* side;

}

},

***RECTANGLE*** {

*@Override*

double area(double... params) {

double length = params[0];

double width = params[1];

return length \* width;

}

},

***TRIANGLE*** {

*@Override*

double area(double... params) {

double base = params[0];

double height = params[1];

return 0.5 \* base \* height;

}

};

abstract double area(double... params);

}

public class AreaCal {

public static void main(String[] args) {

double[][] sampleParams = {

{5},

{4},

{6, 3},

{8, 5}

};

*Shape*[] shapes = *Shape*.*values*();

for (int i = 0; i < shapes.length; i++) {

double area = shapes[i].area(sampleParams[i]);

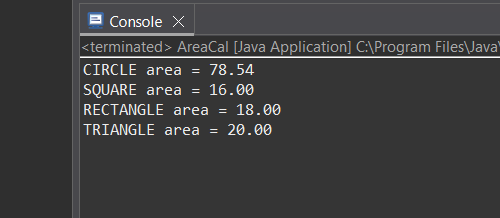
System.***out***.printf("%s area = %.2f%n", shapes[i], area);

}

}

}

**OUTPUT:**



**4.Answer**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

enum *Suit* {

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

}

enum *Rank* {

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

}

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;

}

}

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 class CradsDemo {

public static void main(String[] args) {

Deck deck = new Deck();

System.***out***.println("Original deck:");

deck.printDeck();

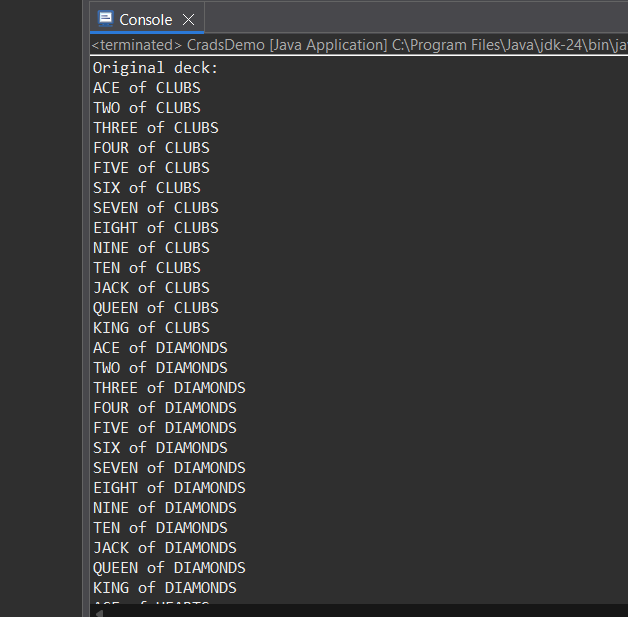
deck.shuffle();

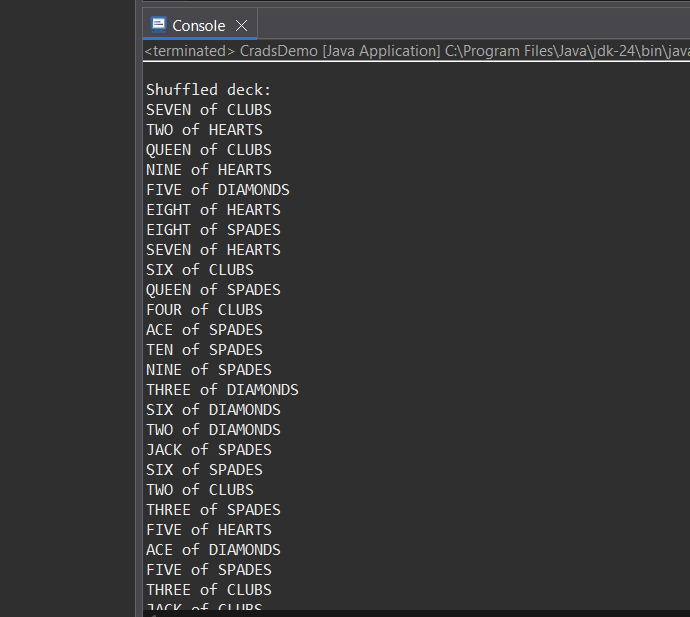
System.***out***.println("\nShuffled deck:");

deck.printDeck();

}

}

**OUTPUT:**



1. **Answer**

public enum Priority {

LOW(1),

MEDIUM(2),

HIGH(3),

CRITICAL(4);

private final int severity;

private static final int URGENCY\_THRESHOLD = 3;

Priority(int severity) {

this.severity = severity;

}

public int getSeverity() {

return severity;

}

public boolean isUrgent() {

return severity >= URGENCY\_THRESHOLD;

}

public String getDescription() {

return String.format("%s (severity: %d) - %s",

this.name(), severity, isUrgent() ? "urgent" : "not Urgent");

}

public static void main(String[] args) {

for (Priority level : Priority.values()) {

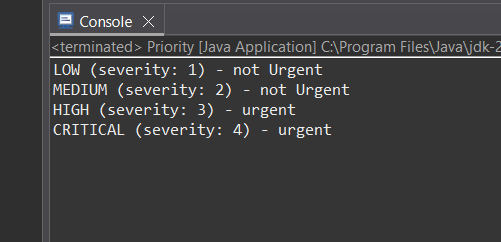
System.out.println(level.getDescription());

}

}

}

**OUTPUT:**



**6.Answer**

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***;

}

};

*@Override*

public String toString() {

return name();

}

}

public class TrafficLights {

public static void main(String[] args) {

State current = *TrafficLight*.***RED***;

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

for (int i = 1; i <= 6; i++) {

current = current.next();

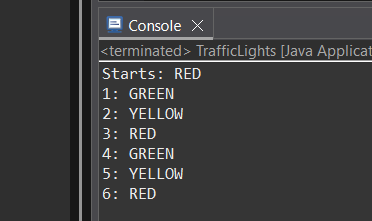
System.***out***.println(i + ": " + current);

}

}

}

**OUTPUT:**



**7.Answer**

enum *Difficulty* {

***EASY***,

***MEDIUM***,

***HARD***

}

class Game1 {

private int bullets;

public Game1(*Difficulty* diff) {

switch (diff) {

case ***EASY***:

bullets = 3000;

break;

case ***MEDIUM***:

bullets = 2000;

break;

case ***HARD***:

bullets = 1000;

break;

default:

bullets = 0;

}

System.***out***.println("difficult: " + diff + " → bullets: " + bullets);

}

}

public class Game {

public static void main(String[] args) {

new Game1(*Difficulty*.***EASY***);

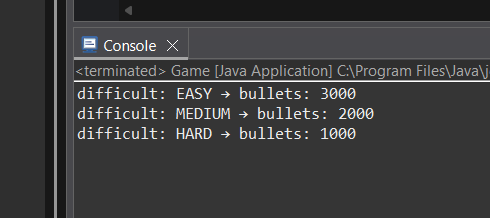
new Game1(*Difficulty*.***MEDIUM***);

new Game1(*Difficulty*.***HARD***);

}

}

**OUTPUT:**



**8.Answer**

enum *OperationSwitch* {

***PLUS***, ***MINUS***, ***TIMES***, ***DIVIDE***;

public double eval(double a, double b) {

switch (this) {

case ***PLUS***:

return a + b;

case ***MINUS***:

return a - b;

case ***TIMES***:

return a \* b;

case ***DIVIDE***:

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

default:

throw new AssertionError("Unknown operation " + this);

}

}

}

enum *OperationOverride* {

***PLUS*** {

*@Override*

public double eval(double a, double b) {

return a + b;

}

},

***MINUS*** {

*@Override*

public double eval(double a, double b) {

return a - b;

}

},

***TIMES*** {

*@Override*

public double eval(double a, double b) {

return a \* b;

}

},

***DIVIDE*** {

*@Override*

public double eval(double a, double b) {

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

}

};

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

}

public class CalculatorDemo {

public static void main(String[] args) {

double a = 10, b = 5;

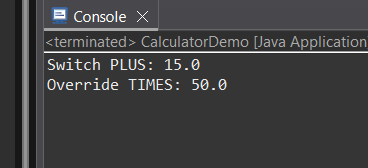
System.***out***.println("Switch PLUS: " + *OperationSwitch*.***PLUS***.eval(a, b));

System.***out***.println("Override TIMES: " + *OperationOverride*.***TIMES***.eval(a, b));

}

}

**OUTPUT:**



**10. Answer**

public enum *KnowledgeLevel* {

*BEGINNER*,

*ADVANCED*,

*PROFESSIONAL*,

*MASTER*;

public static *KnowledgeLevel* fromScore(int score) {

if (score >= 0 && score <= 3) {

return *BEGINNER*;

} else if (score >= 4 && score <= 6) {

return *ADVANCED*;

} else if (score >= 7 && score <= 9) {

return *PROFESSIONAL*;

} else if (score == 10) {

return *MASTER*;

} else {

throw new IllegalArgumentException("Score is from 0 and 10");

}

}

public static void main(String[] args) {

int[] testScores = {0, 3, 4, 6, 7, 9, 10};

for (int score : testScores) {

System.*out*.printf("Score %d → %s%n", score, *fromScore*(score));

}

try {

*fromScore*(-1);

} catch (IllegalArgumentException e) {

System.*out*.println("Exception for -1: " + e.getMessage());

}

try {

*fromScore*(11);

} catch (IllegalArgumentException e) {

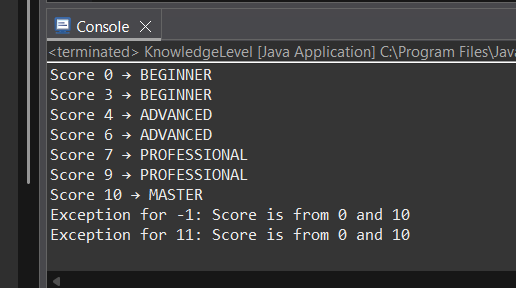
System.*out*.println("Exception for 11: " + e.getMessage());

}

}

}

**OUTPUT:**

****

**Exceptional Handling:**

1. **Answer**

public class ExceptionDemo {

public static void main(String[] args) {

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

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

} finally {

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

}

try {

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

int value = arr[5];

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index is out of bounds");

} finally {

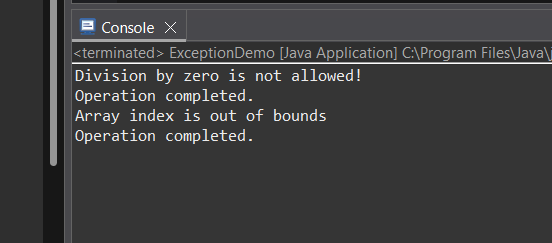
System.out.println("Operation completed.");

}

}

}

**OUTPUT:**



1. **Answer**

public class OddChecker {

public static class OddNumberException extends Exception {

public OddNumberException(String message) {

super(message);

}

}

public static void checkOdd(int n) throws OddNumberException {

if (n % 2 != 0) {

throw new OddNumberException("Odd number: " + n);

}

}

public static void main(String[] args) {

int[] testNumbers = {2, 3, 4, 7, 10};

for (int num : testNumbers) {

try {

*checkOdd*(num);

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

} catch (OddNumberException e) {

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

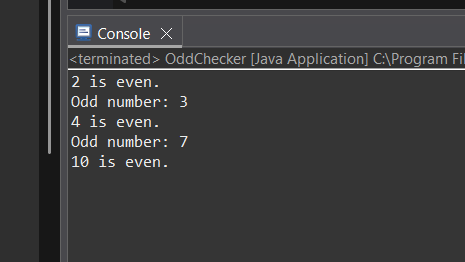
}

}

}

}

**OUTPUT:**



**File Handling**

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileReadDemo {

public static void main(String[] args) {

String filename = "testfile.txt";

try {

*readFile*(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("Cleanup done.");

}

}

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

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

String firstLine = br.readLine();

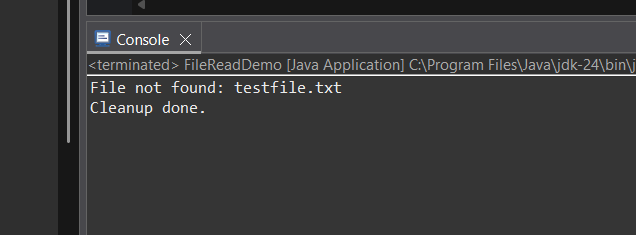
System.***out***.println("First line: " + firstLine);

br.close();

}

}

**OUTPUT:**

****

**4.Multi-Exception**

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class MultiExceptionDemo {

public static void main(String[] args) {

String filename = "numbers.txt";

try {

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

String firstLine = br.readLine();

int number = Integer.*parseInt*(firstLine);

int result = 100 / number;

System.*out*.println("Result: " + result);

br.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 {

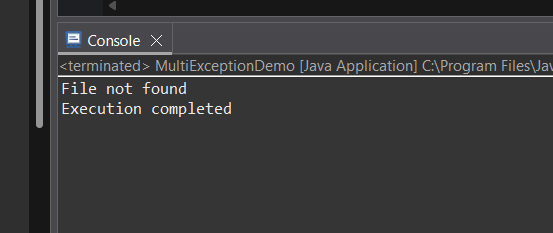
System.*out*.println("Execution completed");

}

}

}

**OUTPUT:**

****