**RECAP:**

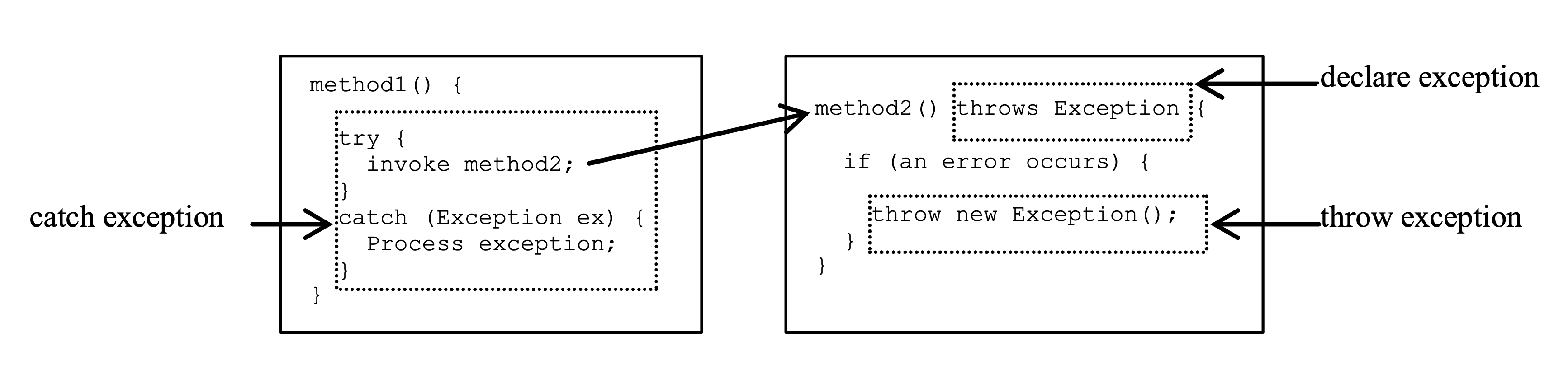
**Record**:

- In real life, different types of data, not uniform data, are stored in information systems

- The record keyword indicates the record definition.

**Exception management:**

- Exceptions are used to manage run-time errors. We use try-catch for that matter.



- throw new TheException();   
TheException ex = new TheException();  
throw ex

- try {

statement;

}

catch (Exception exVar1) {

Handle1;

}

finally {

finalStatement; //This Block always executed.

}

**Files**:

* In Java, files are treated as sequential streams of bytes. Operating systems have mechanisms that determine each file break with an end-of-file-marker.
* Byte-based streams, input and output are in binary format. One char 2 bytes, one int 4 bytes, one double 8 bytes, etc.
* Character-based streams, input and output are in character format. Each character is 2 bytes. The number of bytes depends on the number of characters in a value. For example, 10 bytes for 20000 (5 characters, 2 bytes per character)
* The file created with byte-based streams is called a binary file, the file created with character-based streams is called a text file.
* Each file ends with an end-of-file marker that indicates the end of the file.   
  When a file is opened, a stream is associated with the file.  
  When a program starts running, 3 types of streams are automatically opened:

standard input (System.in),

standard output (System.out)

standard error (System.err)

* the java.io.\* library is used for files

Question 1:

Write a program that uses a record to represent a game character with name, level, and items. Include a method to display the character's details.

public record GameCharacter(String name, int level, String[] items) {  
  
 public void displayDetails() {  
 System.*out*.println("Name: " + name);  
 System.*out*.println("Level: " + level);  
 System.*out*.print("Items: ");  
 for (String item : items) {  
 System.*out*.print(item + " ");  
 }  
 System.*out*.println();  
 }  
}

public static void main(String[] args) {  
 String[] items = {"Sword", "Shield", "Potion"};  
 GameCharacter character = new GameCharacter("Bob", 5, items);  
 character.displayDetails();  
}

Question 2:

Create a record for a bank account with the account holder's name, account number, and current balance. Include methods to deposit and withdraw money from the account.

public class Main {  
 public static void main(String[] args) {  
 BankAccount account = new BankAccount("John Doe", "123456789", 5000.00);  
 account.deposit(2000.00);  
 account.withdraw(10000.00);  
 account.withdraw(3000.00);  
 }  
}  
  
record BankAccount(String name, String accountNumber, double balance) {  
 public void deposit(double amount) {  
 double newBalance = balance + amount;  
 System.*out*.println("Deposited " + amount + " successfully. Current balance: " + newBalance);  
 }  
  
 public void withdraw(double amount) {  
 if (amount > balance) {  
 System.*out*.println("Insufficient funds. Current balance: " + balance);  
 } else {  
 double newBalance = balance - amount;  
 System.*out*.println("Withdrawn " + amount + " successfully. Current balance: " + newBalance);  
 }  
 }  
}

Question 3:

Write a program that throws an exception if a number is divided by zero. Use a try-catch-finally block to handle the exception and display an error message.

import java.util.Scanner;  
  
public class DivideByZeroException {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
 try {  
 System.*out*.print("Enter a number: ");  
 int num1 = input.nextInt();  
 System.*out*.print("Enter another number: ");  
 int num2 = input.nextInt();  
 int result = num1 / num2;  
 System.*out*.println("Result: " + result);  
 } catch (ArithmeticException e) {  
 System.*out*.println("Error: " + e.getMessage());  
 } finally {  
 System.*out*.println("Program completed.");  
 }  
 }  
}

Question 4:

Write a java program that reads a file and throws an exception if the file contains a specific word that the user enters.

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.Scanner;  
  
public class FileSearchException {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.print("Enter file name: ");  
 String fileName = input.nextLine();  
 System.*out*.print("Enter search word: ");  
 String searchWord = input.nextLine();  
 try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 if (line.contains(searchWord)) {  
 throw new Exception("File contains search word.");  
 }  
 }  
 System.*out*.println("Search word not found.");  
 } catch (IOException e) {  
 System.*out*.println("Error reading file: " + e.getMessage());  
 } catch (Exception e) {  
 System.*out*.println("Error: " + e.getMessage());  
 }  
 }  
}

Question 5:

Write a program that reads a file and finds the longest word in the file.

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
  
public class LongestWord {  
 public static void main(String[] args) {  
 String fileName = "example.txt"; // Replace with your file name  
 try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  
 String line;  
 String longestWord = "";  
 while ((line = reader.readLine()) != null) {  
 String[] words = line.split("[^a-zA-Z0-9']+");  
 for (String word : words) {  
 if (word.length() > longestWord.length()) {  
 longestWord = word;  
 }  
 }  
 }  
 System.*out*.println("Longest word: " + longestWord);  
 } catch (IOException e) {  
 System.*out*.println("Error reading file: " + e.getMessage());  
 }  
 }  
}

Question 6:

Create a program that reads a file and encrypts its contents using a Caesar cipher.

Caesar cipher: The Caesar cipher is a simple encryption technique that involves shifting each letter of a message by a certain number of positions in the alphabet. For example, if we shift each letter by 3 positions, the letter A becomes D, B becomes E, C becomes F, and so on. To decrypt the message, we simply shift each letter back by the same number of positions.

import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Scanner;  
  
public class CaesarCipher {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
 System.*out*.print("Enter file name: ");  
 String fileName = input.nextLine();  
 System.*out*.print("Enter shift amount: ");  
 int shift = input.nextInt();  
 input.nextLine(); // Consume newline  
 try (BufferedReader reader = new BufferedReader(new FileReader(fileName));  
 FileWriter writer = new FileWriter("encrypted.txt")) {  
 int c;  
 while ((c = reader.read()) != -1) {  
 if (Character.*isLetter*(c)) {  
 c = (c - 'A' + shift) % 26 + 'A';  
 }  
 writer.write(c);  
 }  
 System.*out*.println("Encryption complete.");  
 } catch (IOException e) {  
 System.*out*.println("Error: " + e.getMessage());  
 }  
 }  
}