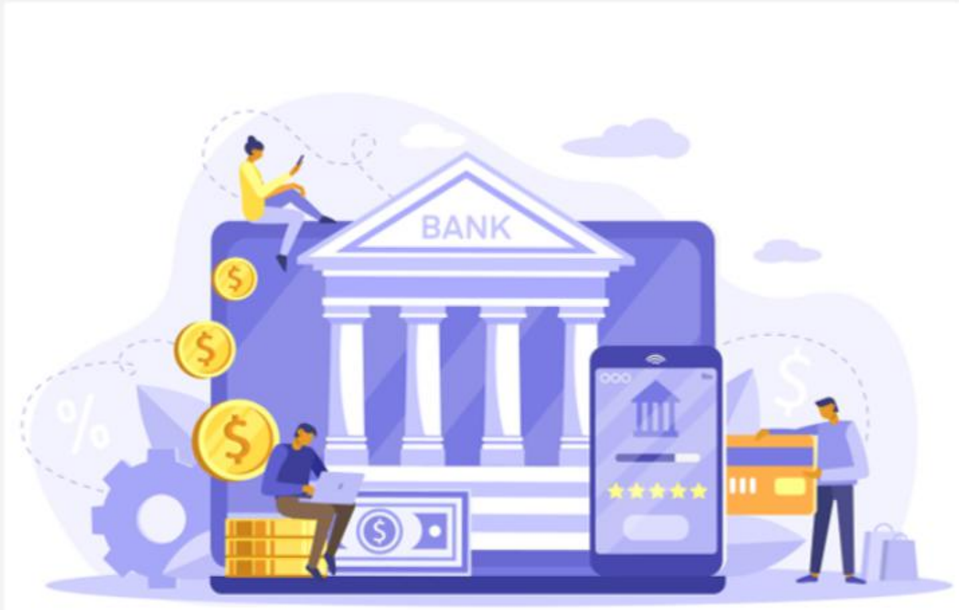
Create and Implement a User-Defined Exception

Automated Grading

- Teachers, along with the technical team at Oak Bridge School, must automate their grading process. The teams must:
  - accept the marks scored by each student in each subject.
  - calculate the total marks scored by each student.
  - evaluate the grades obtained by each student.
- If you were a part of their team, how would you ensure that the marks obtained did not exceed 100 in any of the subjects?

Slide Note



Opening Bank Accounts Online

- The software team at IDBC Bank needs to build an application to help customers open savings and/or checking accounts online.
- As a member of their team, how will you ensure that the account balance is not a negative number?

Slide Note

Can the below scenarios be handled using Exception Handling ?

• If the marks entered for a student is greater than 100, can we write an exception handler for the same?

• If the bank balance goes below zero, can we write an exception handler for it?

Think and Tell

- How can we ensure students' marks do not exceed 100 in any subject or that an account balance is not negative?
- Is there a predefined class that can be used as an exception handler here?
- What will you do if there is no suitable predefined class in the Java exception handling library?

Slide Note

Learning Objectives

- Understand the Checked Exception
- Introduction to User-defined Exception
- Implement Throw and Throws Keyword
- Difference Between Checked and Unchecked Exception

Slide Note

Understand Checked Exceptions

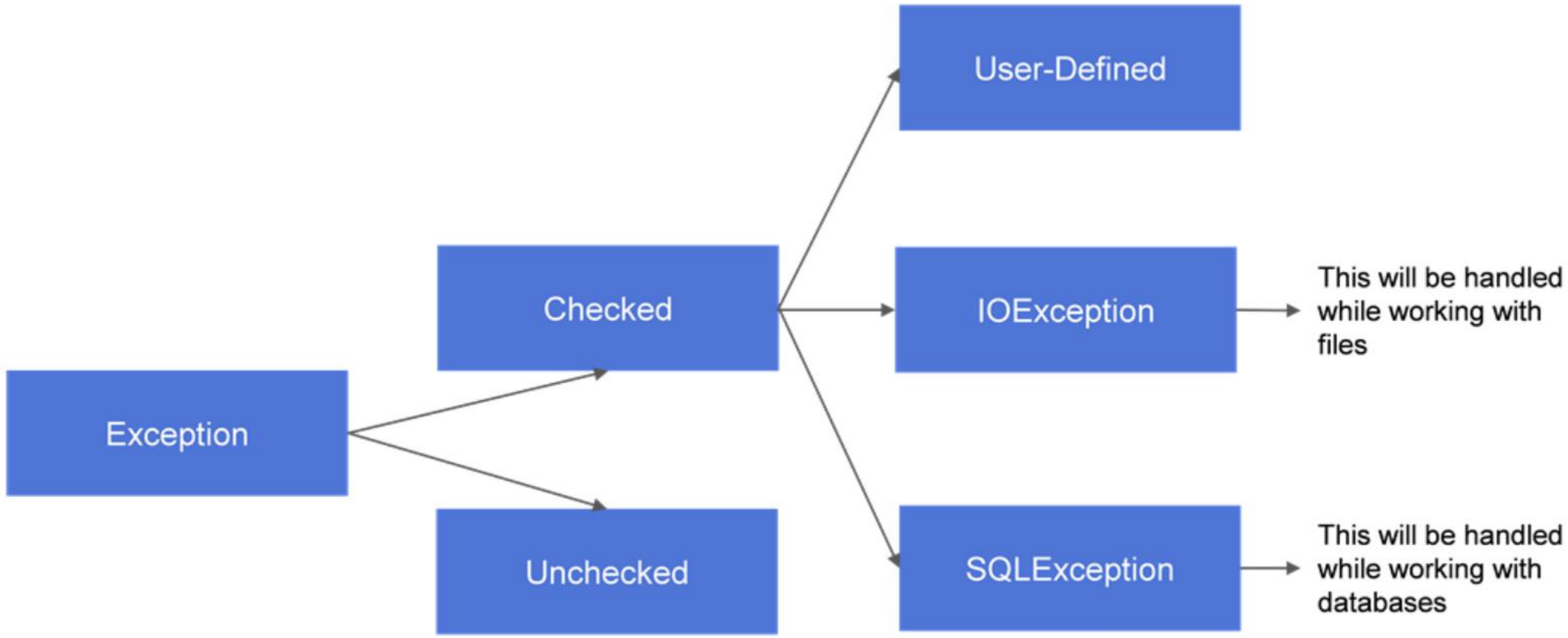
Slide Note

Checked Exceptions

- These are the exceptions that are detected at compile time.
- Checked exceptions are exceptional scenarios that we can anticipate in a program and try to recover from.
- We should catch the exception and provide a meaningful message to the user.
- If a checked exception is thrown:
  - We should either handle it in the same manner as before, or
  - We should propagate it to the method that calls it

Slide Note

Types of Checked Exceptions



Slide Note

Introduction to User-Defined Exceptions

Slide Note

User-Defined Exceptions

- Java allows programmers to create their own exceptions.
  - Such exceptions are known as "custom exceptions" or "user-defined exceptions"
- User-defined exceptions are classes that provide flexibility to add attributes and methods that are not part of a standard Java exception library.
- All user-defined exceptions are checked exceptions.

Slide Note

Creating User-Defined Exceptions

- To create a user-defined exceptions:
  - Create an exception class
  - The created class needs to inherit the `java.lang.Exception` class, as shown in the image.
- The `PlayerNotFoundException` is a user-defined exception class.

```
public class PlayerNotFoundException extends Exception{

    public PlayerNotFoundException(String message) {
        super(message);
    }
}
```

Menu

00:11:00:00:22

11/ 22

Slide Note

Implement the Throw and Throws Keyword

Slide Note

Declaring User-Defined Exceptions

```
public void getAllPlayers(int age) throws PlayerNotFoundException{

    //This method will throw PlayerNotFoundException
    //if player of specific age is not found
}
```

- If a method does not handle a checked exception, the `throws` keyword is used at the end of the method signature. It is called declaring an exception.
- The `throws` keyword indicates the type of exception that is likely to be thrown by a method.
- The `getAllPlayers` method may throw an exception if a player of a specific age is not found.

Slide Note

The Throw Keyword

- User-defined exceptions can be explicitly thrown from a method. It is done using the `throw` keyword.
- The `getAllPlayers` method throws the `PlayersNotFoundException` when the age is less than 40.
- The checked exception is thrown using the `throw` keyword from inside the `getAllPlayers` method.
- Any method that explicitly throws an exception using the `throw` keyword must declare the exception in the method signature using the `throws` keyword.
- Only an object of type exception will be thrown using the throw keyword.
- `PlayerNotFoundException` extends the `Exception` class. So, it is a type of Exception class and can be used with the throw keyword.

```
public static void getAllPlayers(int age, Player player) throws PlayerNotFoundException {

    //if player of specific age is not found throw the exception
    if(player.getAge()<40){
        throw new PlayerNotFoundException("Player of age above 40 is not Found");
    }
}
```

Slide Note

Handling User-Defined Exceptions In a Program

```
public static void getAllPlayers(int age, Player player) throws PlayerNotFoundException {

    //if player of specific age is not found throw the exception
    if(player.getAge()<40){
        throw new PlayerNotFoundException("Player of age above 40 is not Found");
    }
}
public static void main(String[] args) {
    try {
        getAllPlayers( age: 41,player);
    } catch (PlayerNotFoundException e) {
        e.printStackTrace();
    }
}
```

- The `getAllPlayers` method is called from the main method.
- Here, the method `getAllPlayer` is throwing a checked exception using the throws keyword.
- The main method must handle the exception using a try catch block or
- It can propagate it to any other method.
- So, when throwing a checked exception, it must be handled by the calling method using a try. Catch or by declaring using the throws keyword.

Slide Note

The Difference Between Checked and Unchecked Exceptions

Slide Note

Checked vs Unchecked Exceptions

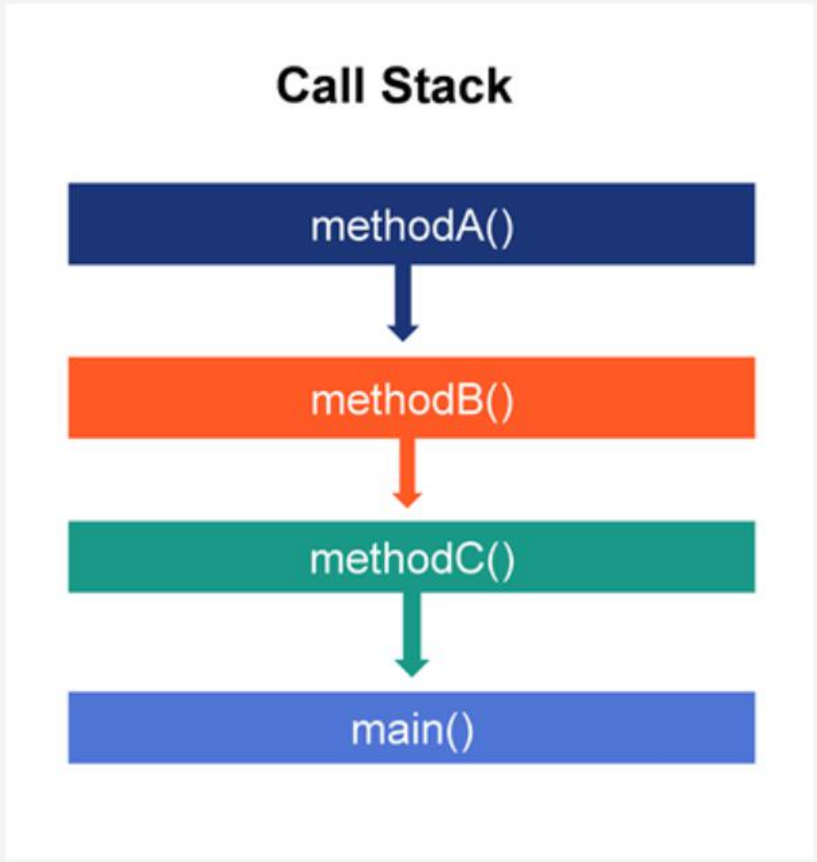
Checked Exceptions	Unchecked Exceptions
They occur at compile time	They occur at runtime
The compiler gives a compile-time error	The compiler does not give a compile-time error
These types of exceptions need to be handled at compile time	These types of exceptions cannot be handled at the time of compilation, because they are generated by logical mistakes in the program
The JVM requires that the exception be caught and handled	The JVM does not require the exception to be caught and handled

Example of Checked Exceptions:
`IOException`
`FileNotFoundException`
`ClassNotFoundException`
	Examples of Unchecked Exceptions:
	`NullPointerException`
	`ArrayIndexOutOfBoundException`
	`ArithmeticException`

Slide Note

Exception Propagation

- In Java, all methods are stacked in the memory in the order in which they are called by the main method.
- The exceptions that occur during execution are propagated down the stack of methods until an appropriate exception handler is encountered.
- The exception handler is usually a try and catch block.



Slide Note

Quick Check

Predict the output for the following code:

```
public static void main(String[] args) {
   try
   {
      int a[]= {1, 2, 3, 4};
      for (int i = 1; i <= 4; i++)
      {
         System.out.println ("a[" + i + "]=" + a[i] + "n");
      }
   }
   catch (Exception e)
   {System.out.println ("error = " + e);}
   catch (ArrayIndexOutOfBoundsException e)
   {System.out.println ("ArrayIndexOutOfBoundsException");}
}
```

Options:

1. Compile time error
2. ArrayIndexOutOfBoundsException
3. Array is printed
4. "error =" Exception

Slide Note

Quick Check: Solution

Predict the output for the following code:

```
public static void main(String[] args) {
    try
    {
        int a[]= {1, 2, 3, 4};
        for (int i = 1; i <= 4; i++)
        {
            System.out.println ("a[" + i + "]=" + a[i] + "n");
        }
    }
    catch (Exception e)
    {System.out.println ("error = " + e);}
    catch (ArrayIndexOutOfBoundsException e)
    {System.out.println ("ArrayIndexOutOfBoundsException");}
}
```

Options:

1. Compile time error
2. ArrayIndexOutOfBoundsException
3. Array is printed
4. "error =" Exception