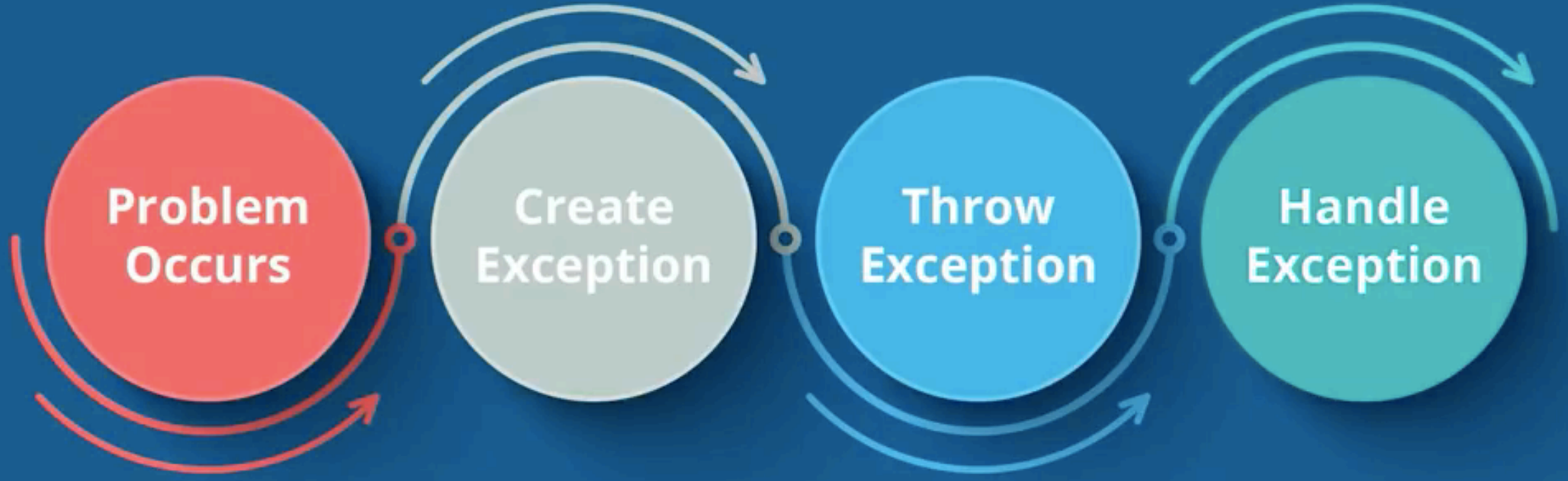
Java-Exception Handling

Topics For Today's Discussion

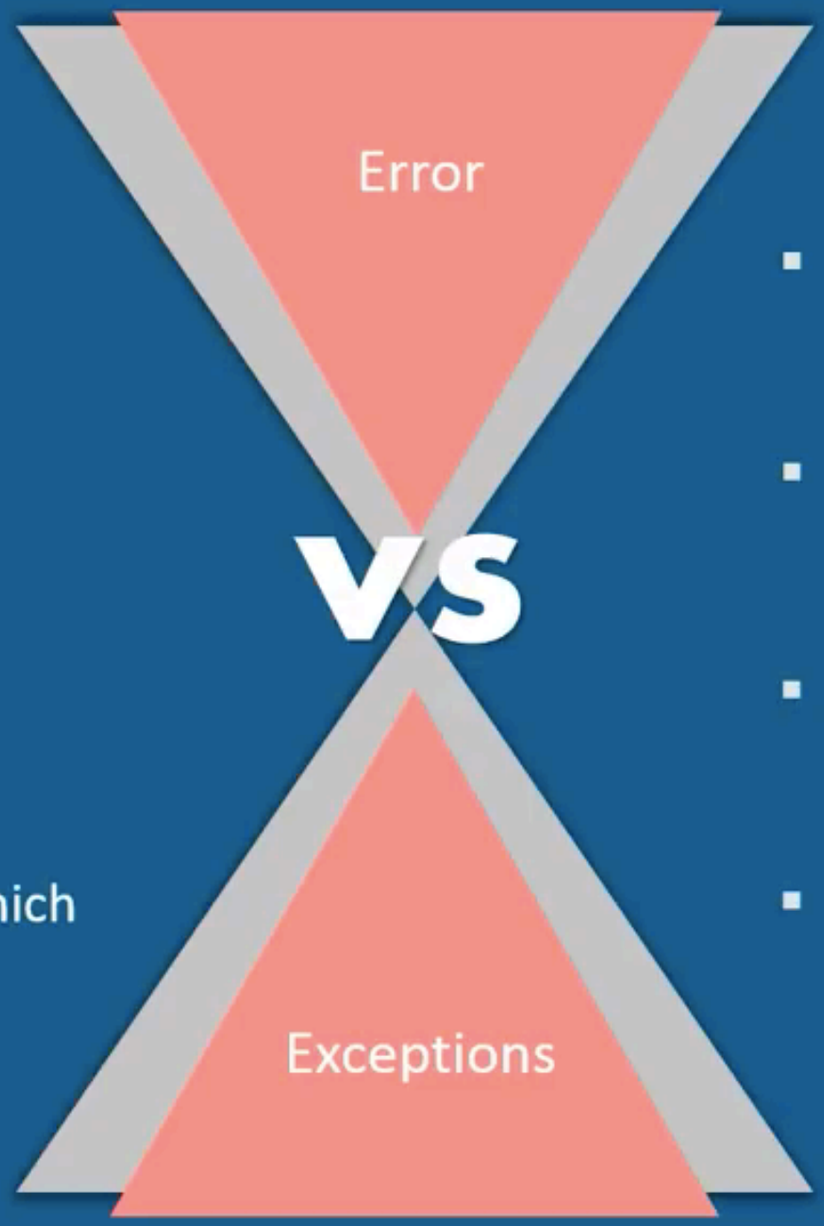
Exception Handling

An exception is an event that disrupts the normal flow of the program. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.



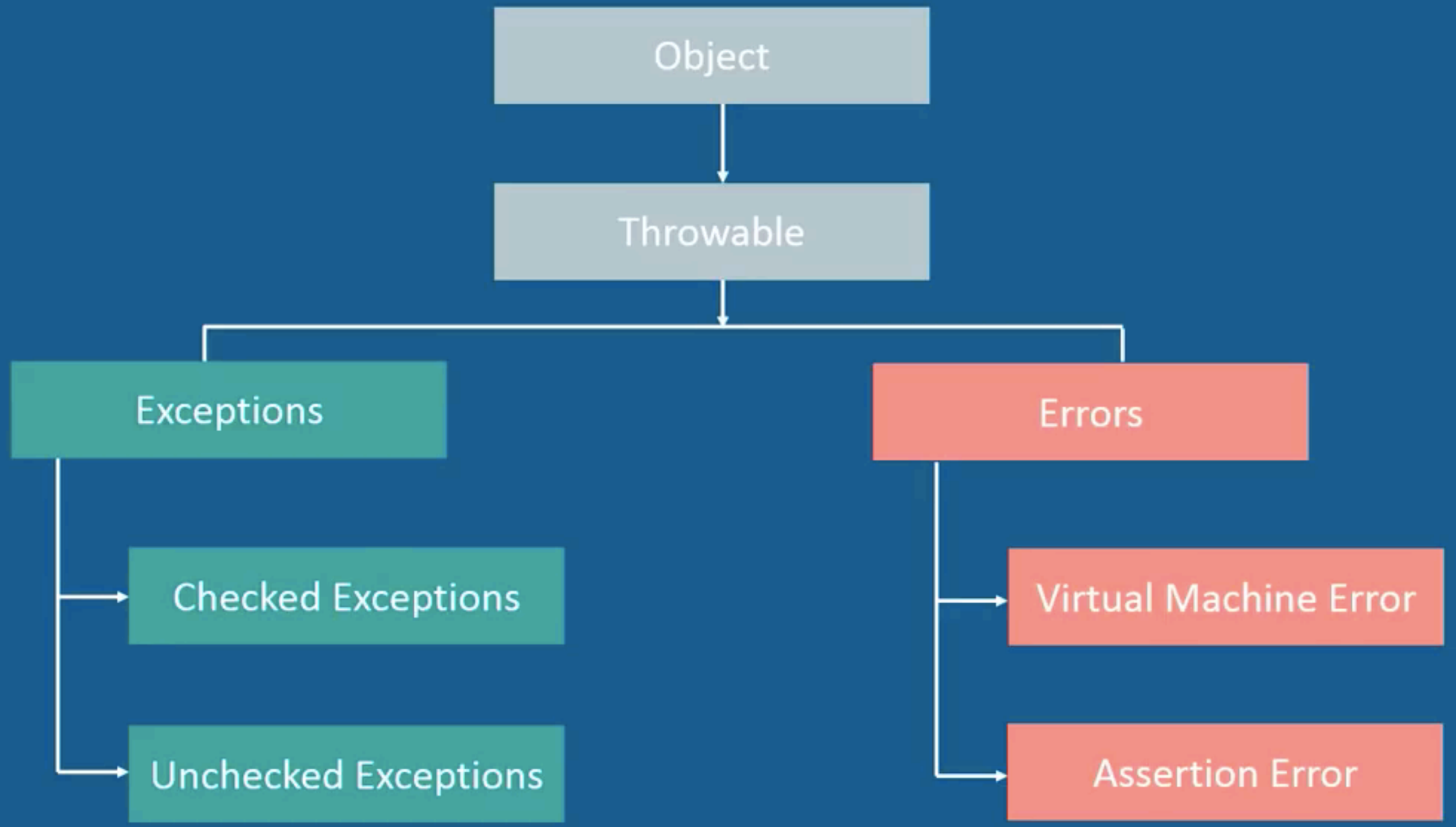
Error vs Exception

- Impossible to recover from error
- Errors are of Unchecked Type
- Happen at Run-Time
- Caused by the environment on which application is running



- Possible to recover from Exception
- It can be of checked or unchecked
- Can happen at Compile Time & Runtime
- Caused by Application

Exception Hierarchy



Types of Exceptions

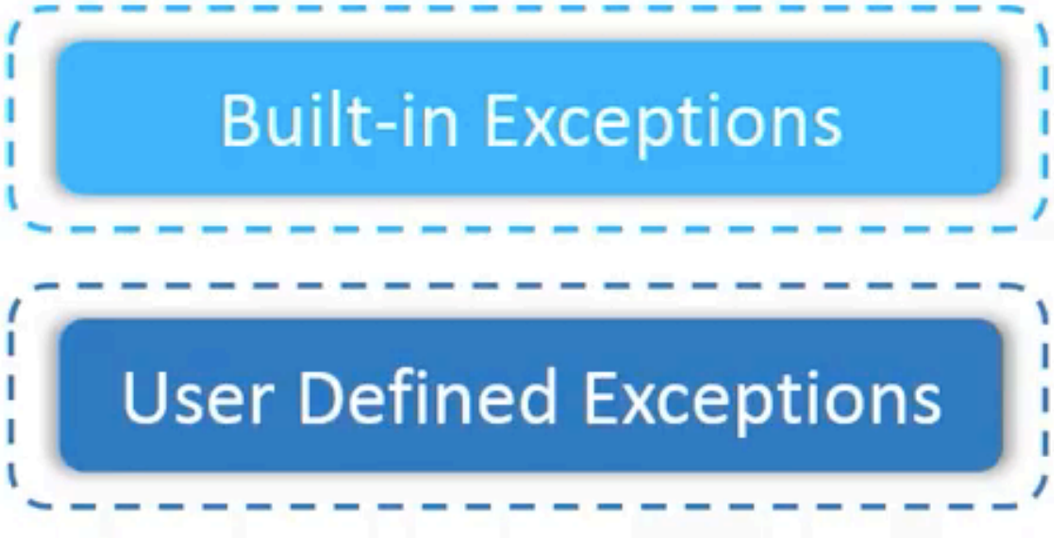
- An exception that is checked by the compiler at compilation-time.
- These exceptions cannot simply be ignored, the programmer should handle these exceptions.

- An exception that occurs at the time of execution.
- These are also called as **Runtime Exceptions**.
- Runtime exceptions are ignored at the time of compilation.

Basic Example format of Exception

```
class Exception{
    public static void main(String args[]){
        try{
            //code that may raise exception
            }
        catch(Exception e){
            // rest of the program
            }
    }
}
```

Types of Java Exceptions



Built In Exceptions

1. ArithmeticException
2. ArrayIndexOutofBoundException
3. ClassNotFoundException
4. IOException
5. InterruptedException
6. NoSuchFieldException
7. NoSuchMethodException
8. NumberFormatException
9. Runtime Exception
10. StringIndexOutOfBoundsException

Exception Handling Methods

try

catch

finally

throw

throws

Used to specify a block where we should place exception code.

Syntax:

```
try{
//code that throws exception
}catch(Exception_class_Name){}
```

Exception Handling Methods

try

catch

finally

throw

throws

```
try{
try{
    //statements to execute
    }catch(ArithmeticException e)
       {System.out.println(e);}
   try{
      int a[]=new int[5];  a[5]=4;
      }
}
```

Exception Handling Methods

try

catch

finally

throw

throws

Used to handle the exception.

Syntax:

```
try {// Protected code}
catch(ExceptionName e1)
 // Catch block}
```

Exception Handling Methods

try

catch

finally

throw

throws

Used to execute the important code of the program.

```
try
  {// Protected code}
catch(Exception e1)
  {// Catch block}
catch(Exception e2)
  {// Catch Block}
finally{
// This block is always executed
}
```

Exception Handling Methods

try

catch

finally

throw

throws

Used to throw an exception.

Syntax:

```
void a(){
throw new ArithmeticException("Incorrect");}
```

```java

class SampleException {
    static void  avg() throws ArithmeticException
    {
        System.out.println("Inside avg function");
        throw new ArithmeticException("Demo");
    }

    public static void main(String args[])
    {
        try {
        avg();
        }
        finally
        {
            System.out.println("Caught");
        }
    }
    }

```

Throw vs Throws

1. Used to explicitly throw an Exception.
2. Checked Exceptions cannot be propagated using throw only.
3. Followed by an instance.
4. Used within a method.
5. Cannot throw multiple exceptions.

1. Used to declare an Exception.
2. Checked Exceptions can be propagated.
3. Followed by a class.
4. Used with a method Signature.
5. Can declare Multiple Exceptions.

Final vs Finally vs Finalize

1. Keyword

2. Applies restrictions on class, method and variable.

3. final class cant be inherited, method cant be overridden & the variable value cant be changed

1. Block

2. Used to place an important code

3. It will be executed whether the exception is handled or not.

1. Method

2. Used to perform clean-up processing just before the object is garbage collected

```
2
3  class SampleException {
4      public static void main(String args[])
5      {
6          try {
7              throw new MyException(5);
8          }
9          catch(Exception e)
10         {
11             System.out.println(e);
12         }
13     }
14 }
15 class MyException extends Exception
16 {
17     int a;
18     MyException(int b){
19         a=b;
20     }
21     public String toString() {
22         return("Exception number = "+a);
23     }
24 }
```

```java
class SampleException {
  static void validateInput(int number) throws InvalidInputException{
      if(number>100)
      {
          throw new InvalidInputException("Exception");
      }
      System.out.println("Valid Input");
  }
  public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Enter a number less than 100 :  ");
      int number = scanner.nextInt();
      try{
          validateInput(number);
      }catch(InvalidInputException e){
          System.out.println("Caught Exception - Input is greater than 100");
      }
  }
}

class InvalidInputException extends Exception{
  InvalidInputException(String exceptionText){
      super(exceptionText);
  }
}
```

```java
import java.util.*;
import java.text.SimpleDateFormat;

public class SampleException {
    static void convertDateFormat(String inputDate) {
        try {
            SimpleDateFormat sdf = new SimpleDateFormat("dd/mm/yyyy");
            Date date = sdf.parse(inputDate);
            SimpleDateFormat outputsdf = new SimpleDateFormat("yyyy-MM-dd");
            String outputDate = outputsdf.format(date);
            System.out.println("After changing date format to yyyy/MM/dd : "+outputDate);
        }catch(java.text.ParseException e){
            System.out.println("Some error occurred while converting date formats. Exceptio
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter date in dd/MM/yyyy format: ");
        String inputDate = scanner.nextLine();
        convertDateFormat(inputDate);
    }
}
```