# Lecture 10: Exceptions and Exception Handling

CSC 1214: Object-Oriented Programming

# Outline

- Exceptions
- Exception Handling
- Defining Own Exceptions

# Outline

- Exceptions
- Exception Handling
- Defining Own Exceptions

# Exceptions

- An ***exception*** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions

- In Java, an exception is represented as an ***exception object***. An **exception object** (or simply an ***exception***) is an object that describes an unusual or erroneous situation.

- An exception object, contains information about the error, including its type and the state of the program when the error occurred

- In Java, many runtime errors are represented as exceptions that can be handled and dealt with accordingly.

# Exceptions

- Java has a predefined set of exceptions and errors that can occur during execution
- A program can deal with an exception in one of three ways:
  - Ignore it
  - Handle it where it occurs
  - Handle it an another place in the program
- The manner in which an exception is processed is an important design consideration

# Exception Handling

- If an exception is ignored (*not caught*) by the program, the program will terminate abnormally and produce an appropriate message. *This is what we have been doing until now.*
- The message includes a call stack trace that indicates the line on which the exception occurred
- The call stack trace also shows the method call trail that lead to the attempted execution of the offending line

# Example

```
public class DivideByZero {
  public static void main (String[] args) {
    int numerator = 40;
    int denominator = 0;

    System.out.println (“Answer ”+numerator / denominator);
    System.out.println ("This will not be printed.");
  }
}
```

# Example

```
public class DivideByZero {
  public static void main (String[] args) {
    int numerator = 40;
    int denominator = 0;

    System.out.println (“Answer ”+numerator / denominator);
    System.out.println ("This will not be printed.");
  }
}
```

# Output

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivideByZero.main(DivideByZero.java:5)
```

**Type of the exception**

**The line number where the exception occurred**

# Example

```
public class DivideByZero2 {
  public static void main (String[] args) {
    int numerator = 10;
    int denominator = 0;

    method1(numerator, denominator);
    System.out.println ("This will not be printed.");
  }

  static void method1(int num, int denom) {
     method2(num, denom);
     System.out.println ("This will not be printed either.");
  }
  static void method2(int num, int denom) {
      int div = num / denom;
      System.out.println ("no chance here as well");
  }
```

# Example

```
public class DivideByZero2 {
  public static void main (String[] args) {
    int numerator = 10;
    int denominator = 0;

    method1(numerator, denominator);
    System.out.println ("This will not be printed.")
  }

  {

  inted either.")

  }
  sta

  }
}
```

# Output

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivideByZero2.method2(DivideByZero2.java:15)
    at DivideByZero2.method1(DivideByZero2.java:11)
    at DivideByZero2.main(DivideByZero2.java:6)
```

**Type of the exception**

**The method call trail that lead to the attempted execution of the offending line**

# Outline

- Exceptions
- Exception Handling
- Defining Own Exceptions

# The try Statement and the catch Clause

Exception handling:

- To handle an exception when it occurs, the line that throws the exception is executed within a **try** block
- A **try** block is followed by one or more **catch** clauses, which contain code to process an exception
- Each catch clause has an associated exception type and is called an exception handler
- When an exception occurs, processing continues at the first catch clause that matches the exception type

# The try Statement and the catch Clause

## Exception handling

```
try {

  }
catch (ExceptionType name) {

   }
catch (ExceptionType name) {

 }
```

You associate exception handlers with a **try** block by providing one or more **catch** blocks directly after the **try** block. No code can be between the end of the **try** block and the beginning of the first catch block

# The try Statement and the catch Clause

```
class DivideByZero3 {
    public static void main(String args[]){
        int numerator = 40;
        int denominator = 0;
        try{
          System.out.println("Answer "+numerator/denominator);
        }
        catch(ArithmeticException e){
          System.out.println("Attempted to divide to Zero");
        }
        System.out.println("This will be printed");
    }
}
```

# The finally Clause

- A **try** statement can have an optional clause following the **catch** clauses, designated by the reserved word **finally**
- The statements in the **finally** clause always are executed
- If no exception is generated, the statements in the **finally** clause are executed after the statements in the **try** block complete
- If an exception is generated, the statements in the **finally** clause are executed after the statements in the appropriate **catch** clause complete

# The finally Clause

```
try {

  }
catch (ExceptionType name) {

   }
catch (ExceptionType name) {

 }
finally{
}
```

# The finally Clause Example

```
class DivideByZero3 {
    public static void main(String args[]){
        int numerator = 40;
        int denominator = 0;
        try{
          System.out.println("Answer "+numerator/denominator);
        }
        catch(ArithmeticException e){
          System.out.println("Attempted to divide to Zero");
        }
        finally{
          System.out.println("This will be printed");
        }
}
```

# Outline

- Exceptions
- Exception Handling
- Defining Own Exceptions

# Defining Own Exceptions

- A programmer can define a custom exception by extending the **Exception** class or one of its descendants

```
class DivideByZeroException extends Exception
{
    // A constructor to initialise the exception object
    // with a particular message.
    DivideByZeroException(String message)
    {
        super(message);
    }
}
```

# The throw Statement

- Exceptions are thrown using the **throw** statement
- Usually a **throw** statement is nested inside an **if** statement that evaluates the condition to see if the exception should be thrown

```
import java.util.Scanner;
public class DivideByZero4 {
    public static void main(String args[]) throws DivideByZeroException{
            int numerator;
            int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            System.out.print("Enter numerator:");
            numerator = scan.nextInt();
            System.out.print("Enter denominator:");
            denominator = scan.nextInt();
            if(denominator == 0)
                throw new DivideByZeroException("Zero Divisor");
            System.out.println("Answer "+numerator/denominator);
            break;
        }
    }
}
```

# The throw Statement

```
import java.util.Scanner;
public class DivideByZero4 {
    public static void main(String args[]) throws DivideByZeroException{
            int numerator;
            int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            System.out.print("Enter numerator:");
            numerator = scan.nextInt();
            System.out.print("Enter denominator:");
            denominator = scan.nextInt();
            if(denominator == 0)
                throw new DivideByZeroException("Zero Divisor");
            System.out.println("Answer "+numerator/denominator);
            break;
        }
```

## Output

```
Enter numerator:10
Enter denominator:0
Exception in thread "main" DivideByZeroException: Zero Divisor
    at DivideByZero4.main(DivideByZero4.java:24)
```

# The throws Clause

- The main method of the *DivideByZero4* class has a **throws** clause, indicating that it may throw an *DivideByZeroException*. The **throws** clause is required because the *DivideByZeroException* was derived from the Exception class, making it a **checked** exception.

```
import java.util.Scanner;
public class DivideByZero4 {
    public static void main(String args[]) throws DivideByZeroException{
            int numerator;
            int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            System.out.print("Enter numerator:");
            numerator = scan.nextInt();
            System.out.print("Enter denominator:");
            denominator = scan.nextInt();
            if(denominator == 0)
                throw new DivideByZeroException("Zero Divisor");
            System.out.println("Answer "+numerator/denominator);
            break;
        }
    }
}
```

# Custom Exceptions are also Handled Using the try and catch clauses

```
import java.util.Scanner;
public class DivideByZero5 {
   public static void main(String args[]) throws DivideByZeroException{
         int numerator;
         int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            try{
                System.out.print("Enter numerator:");
                numerator = scan.nextInt();
                System.out.print("Enter denominator:");
                denominator = scan.nextInt();
                if(denominator == 0)
                    throw new DivideByZeroException("Zero Divisor");
                System.out.println("Answer "+numerator/denominator);
                break;
            }
            catch(DivideByZeroException exception){
                System.out.println("Attempted to divide by zero please try again");
                System.out.println("Exception details: "+exception.getMessage());
            }
        }
   }
}
```

# Custom Exceptions are also Handled Using the try and catch Clauses

```
import java.util.Scanner;
public class DivideByZero5 {
      public static void main(String args[]) throws DivideByZeroException{
                int numerator;
                int denominator;
         Scanner scan = new Scanner(System.in);
         while(true){
            try{
                System.out.print("Enter numerator:");
                numerator = scan.nextInt();
                System.out.print("Enter denominator:");
                denominator = scan.nextInt();
                if(denominator == 0)
                    throw new DivideByZeroException("Zero Divisor");
                System.out.println("Answer "+numerator/denominator);
                break;
            }
            catch(DivideByZeroException exception){
                System.out.println("Attempted to divide by zero please try again");
                System.out.println("Exception details: "+exception.getMessage());
            }
         }
      }
}
```

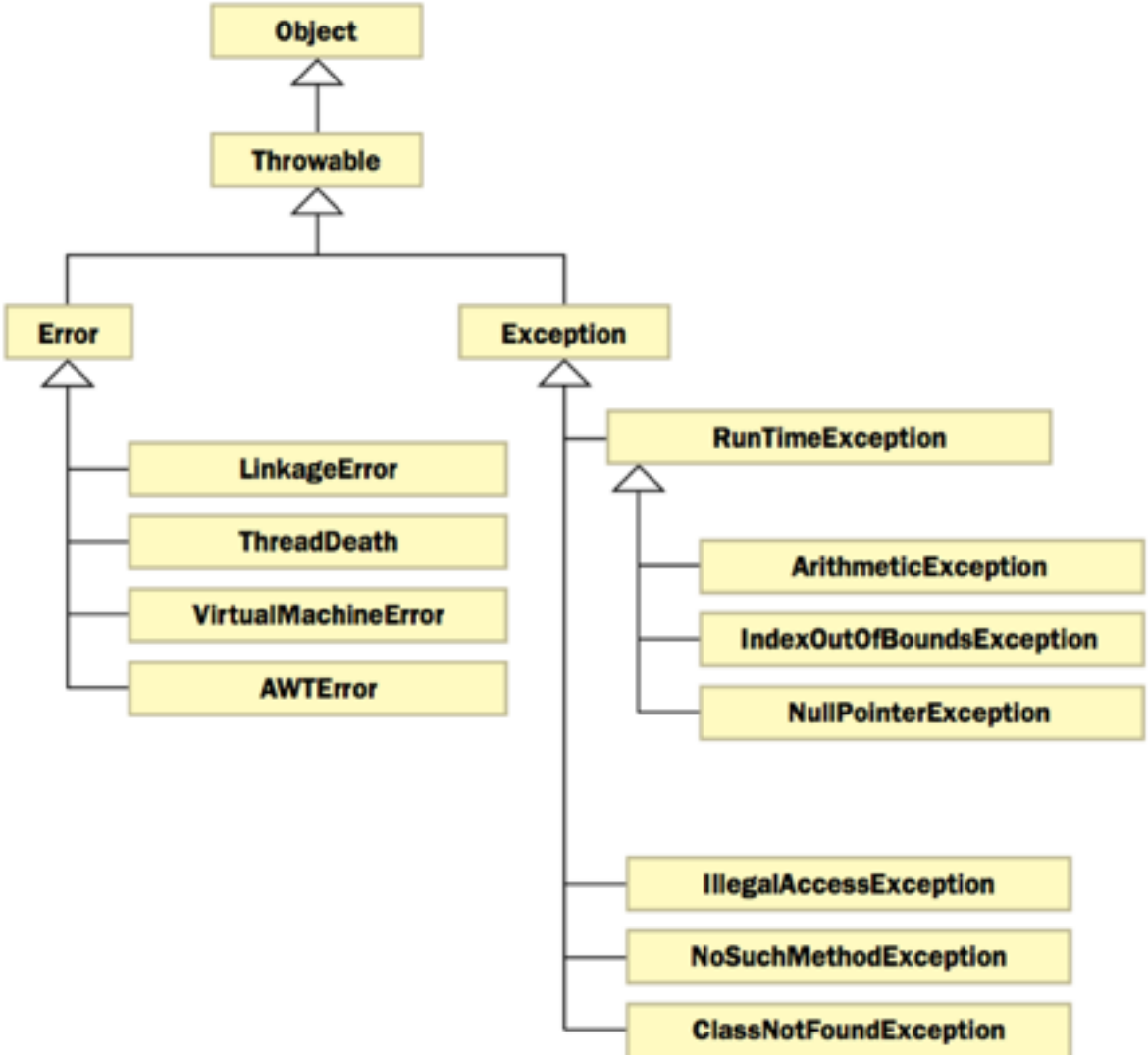
Output

```
Enter numerator:10
Enter denominator:0
Attempted to divide by zero please try again
Exception details: Zero Divisor
Enter numerator:8
Enter denominator:4
Answer 2
```

# Checked Vs. Unchecked Exceptions

- An exception is either **checked** or **unchecked**
- A **checked exception** must either be caught <u>or</u> must be listed in the throws clause of any method that may throw or propagate it
- A throws clause is appended to the method header
- The compiler will issue an error if a checked exception is not caught <u>or</u> listed in a throws clause
- An **unchecked** exception does not require explicit handling, though it could be processed that way
- The only unchecked exceptions in Java are objects of type **RuntimeException** or any of its descendants
- Errors are similar to **RuntimeException** and its descendants

# The Exception Class Hierarchy



# Quiz (1)

- What is the problem with this code?

```
import java.util.Scanner;
public class DivideByZero4 {
   public static void main(String args[]){
        int numerator;
        int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            System.out.print("Enter numerator:");
            numerator = scan.nextInt();
            System.out.print("Enter denominator:");
            denominator = scan.nextInt();
            if(denominator == 0)
              throw new DivideByZeroException("Zero Divisor");
            System.out.println("Answer "+numerator/denominator);
            break;
        }
   }
}
```

# Quiz (2)

- What is the problem with this code?

```
import java.util.Scanner;
public class DivideByZero4 {
   public static void main(String args[]) throws DivideByZeroException{
        int numerator;
        int denominator;
        Scanner scan = new Scanner(System.in);
        while(true){
            System.out.print("Enter numerator:");
            numerator = scan.nextInt();
            System.out.print("Enter denominator:");
            denominator = scan.nextInt();
            throw new DivideByZeroException("Zero Divisor");
            System.out.println("Answer "+numerator/denominator);
            break;
        }
   }
}
```