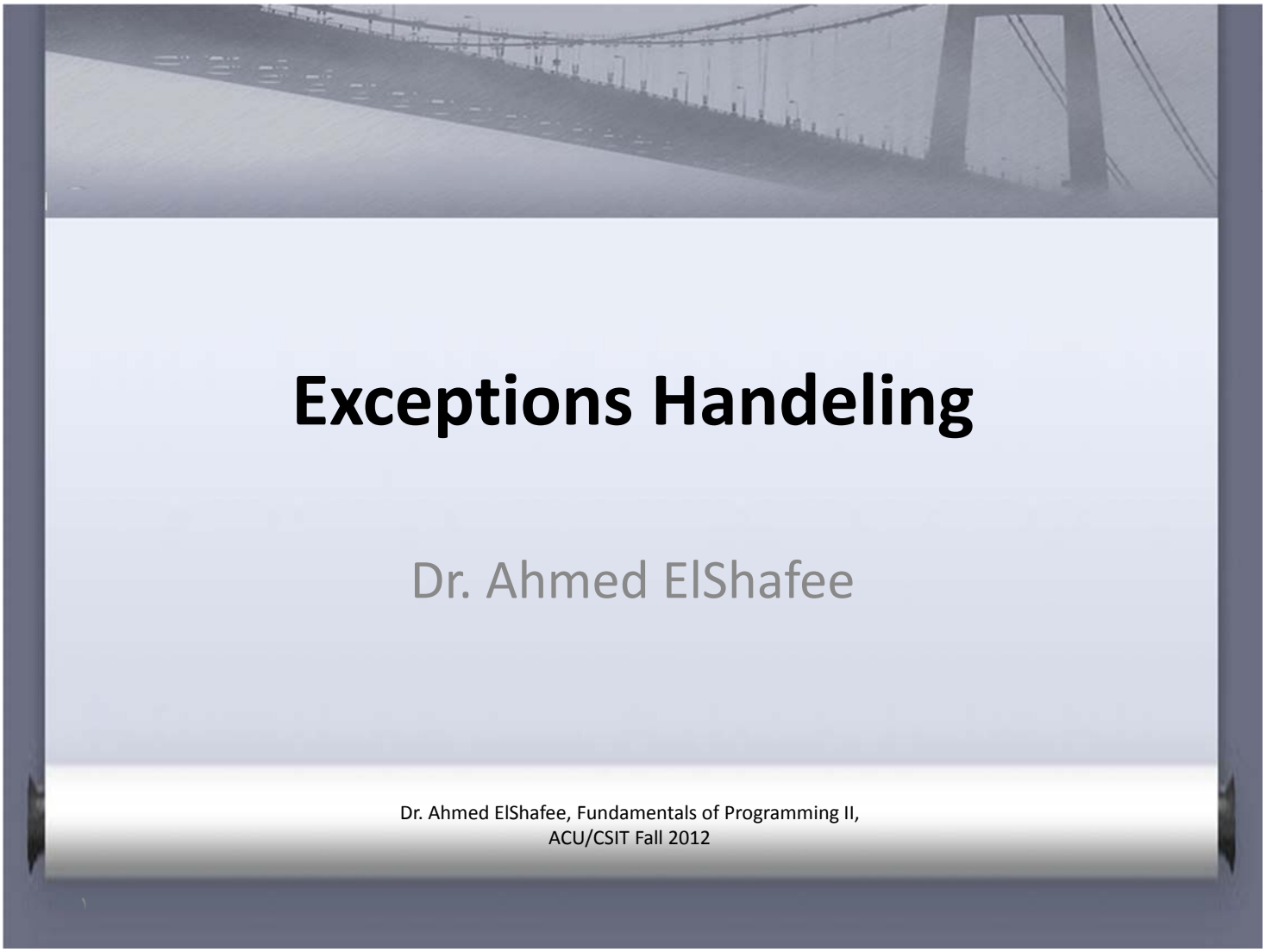

Exceptions Handeling

Dr. Ahmed ElShafee

Agenda

1. *
2. *
3. *

Introduction

- During the execution of a program, the computer will face two types of situations: those it is prepared to deal with and those it is not.
- Imagine you write a program that requests a number from the user:

```
import java.util.Scanner;
public class Exercise {
public static void main(String[] args) {
double side;
Scanner scnr = new Scanner(System.in);
System.out.println("Square Processing");
System.out.print("Enter Side: ");
side = scnr.nextDouble();
System.out.println("\nSquare Characteristics");
System.out.printf("Side: %.2f\n", side);
System.out.printf("Perimeter: %.2f\n", side * 4);}}
```

- This is a classic easy program. When it comes up, the user is asked to simply type a number.
- The number would then be multiplied by 4 and display the result.

```
Square Processing
Enter Side: 24.95
Square Characteristics
Side: 24.95
Perimeter: 99.80
```

- Imagine that a user types something that is not a valid number, such as the name of a
- country or somebody's telephone number. Since this program was expecting a number and it is not prepared to multiply a string to a number, it would not know what to do.

- The only alternative the compiler would have is to send the problem to the operating system, hoping that the OS would know what to do.
- What actually happens is that, whenever the compiler is handed a task, it would try to perform the assignment.
- If it cannot perform the assignment, for any reason it is not prepared for, it would produce an error.
- As a programmer, if you can anticipate the type of error that could occur in your program, you can identify the error yourself and deal with it by telling the compiler what to do when this type of error occurs.

Lecture0801

- Check lab manual

Lecture0802

- Check lab manual

trying

- An exception is an unusual situation that could occur in your program.
- As a programmer, you should anticipate any abnormal behavior that could be caused by the user entering wrong information that could otherwise lead to unpredictable results.
- The ability to deal with a program's abnormal behavior is called exception handling.
- Java provides many keywords and built-in classes to handle an exception.

- To start handling an exception, create a section of code that includes the normal flow of the program.
- This section starts with the **try** keyword followed by an opening curly bracket and a closing curly bracket.
- The section in the curly brackets is the body. In it, you write the normal code.

```
public static void main(String[] args) {
double side;
Scanner scnr = new Scanner(System.in);
try {
System.out.println("Square Processing");
System.out.print("Enter Side: ");
side = scnr.nextDouble();
System.out.println("\nSquare Characteristics");
System.out.printf("Side: %.2f\n", side);
System.out.printf("Perimeter: %.2f\n", side * 4);}}
```

Catching

- Just after the try section, that is, after the closing curly bracket of try, you create a new section that uses the **catch** keyword.

```
catch(ExceptionClass) { WhatToDo }
```

- As you can see, this section starts with the **catch** keyword with parentheses.
- The parentheses behave like those of a method. You must pass an argument.
- At a minimum, you can pass a class named **Exception** and a name for the argument.
- After the closing the parenthesis, create a body for the section.
- Like a normal body, this starts with an opening curly bracket and ends with a closing curly bracket.

- Based on this, the basic formula of exception handling is:

```
try {
// Try the program flow
}
catch(Exception arg) {
// Catch the exception
}
```


Lecture0803

- Check lab manual

Lecture0804

- Check lab manual

Do something with caught error

- As mentioned already, if an error occurs when processing the program in the try section, the compiler transfers the processing to the next catch section.
- You can then use the catch section to deal with the error. At a minimum, you can display a message to inform the user.

Lecture0805

- Check lab manual

Lecture0806

- Check lab manual

Lecture0807

- Check lab manual

Exception Throwing

- Imagine you write a program that requests a student's age from the user.
- As we know, everybody's age is positive. Therefore, we need to figure out what to do if the user types a negative number.
- The expression that checks whether the number entered is positive can be written as

```
if(studentAge < 0)
```

Lecture0808

- Check lab manual

- To throw an exception, you use the **throw** keyword.
- The primary formula of using this keyword is:

```
try {
Normal flow
When to throw the exception
throw What?
}
catch(Exception e) {
// Deal with the exception here
}
```

- The new keyword in this formula is **throw**.
- On the right side of throw, indicate to the compiler what to do.
- This means that the throw keyword is followed by an expression.

- In our introduction to exceptions, we got acquainted with the Exception class.
- Besides the default constructor, the **Exception** class is equipped with another constructor that takes a string as argument.
- When using the **throw** keyword, you can use this constructor to specify a message to display to display in case of an error.
- To do this, use the **new** operator to call the constructor and pass a message to it.


```
public class Exercise {
public static void main(String[] args) {
      int studentAge = 0;
      Scanner scnr = new Scanner(System.in);
      try {
            System.out.print("Student Age: ");
            studentAge = scnr.nextInt();
            if( studentAge < 0 )
                  throw new Exception("Positive Number Required");
            System.out.println("Student Age: " + studentAge);
            }
      catch(Exception exc)
      {
            System.out.println("Error: " + exc.getMessage());
      }
      }
}
```

- If the user enters an invalid value, the program examines the **throw** keyword.
- This **throw** appears to display a string. The compiler registers this string and since there was an exception, the program exits the **try** block (it gets out of the try block even if the rest of the try block is fine) and looks for the first **catch** block it can find.
- If it finds a **catch** that does not take an argument, it would still use the **catch**.
- Otherwise, you can use the **catch** block to display the error string that was sent by the **throw** keyword.

Lecture0809

check lab manual

Lecture0810

- Check lab manual

Catching Multiple Exceptions

- The programs we have seen so far dealt with a single exception. Most of the time, a typical program will use different types of errors.
- Fortunately, you can deal with as many errors as possible.
- To do this, you can create a **catch** block for each (type of) error

```
try {
        Code to Try
}
catch(Arg1)
{
        One Exception
}
catch(Arg2)
{
        Another Exception
}
```

1. The compiler enters the **try** block and follows the normal flow of the program

2. If no exception occurs in the **try** block, the rest of the **try** block is executed.

3. If an exception occurs in the **try** block, the **try** displays a **throw** expression that specifies the type of error that happened.

   a. The compiler gets out of the **try** block and examines the first **catch**

   b. If the first catch does not match the **throw**n error, the compiler switches to the next **catch**.

   This continues until the compiler finds a **catch** that matches the **throw**n error.

c. If one of the **catch**es matches the thrown error, its body executes.

d. If no **catch** matches the thrown error, you have (or the compiler has) two alternatives.

If there is

- no **catch** that matches the error (which means that you did not provide a matching catch), the compiler hands the program flow to the operating system.
- Another alternative is to include a **catch** whose argument is the Exception default constructor.

- The **catch(Exception)** is used if no other **catch**, provided there was another, matches the thrown error.
- The **catch(Exception)**, if included as part of a **catch** clause, must always be the last **catch**, unless it is the only **catch** of the clause.

- Multiple **catch**es are written if or when a try block is expected to throw different types of errors.
- Imagine a program that requests some numbers from the user and performs some operation on the numbers.


```
byte studentAge = 0;
        Scanner scnr = new Scanner(System.in);
        try {
            System.out.print("Student Age: ");
            studentAge = scnr.nextByte();
            if (studentAge < 0) {
                throw new Exception("Positive Number Required");
            }
            System.out.println("Student Age: " + studentAge);
        } catch (InputMismatchException exc) {
            System.out.println("The operation could not be carried because "
                    + "the number you typed is not valid");
        } catch (Exception exc) {
            System.out.println("Error: " + exc.getMessage());
        }
```

Lecture0811

- Check lab manual

Lecture0812

- Check lab manual

Nesting Exceptions

- Like a conditional statement or a class, you can create one exception inside of another

```
Try{
        statment,;
        throw new Exception(“ message”);
        Try
       {
              statement;
              throw new exception(“message”);
       }
       Cacth (Excpetion e)
       {
              System.Out.print(e.message());
       }
}
Catch (Exception e){
              System.out.print(e.message());
}
```


Lecture0813

- Check lab manual

Exceptions and Methods

- One of the ways you can use methods in exception handling is to have a central method that dispatches assignments to other methods.
- The other methods can use the values of variables; for example they can test.
- If an exception occurs, the other method can display a message or throw an exception.
- This throwing can be picked up by the central method that sent the variable.
- The method that originated the trying can hand it to a catch block or one of the catch blocks in the calling method that can handle the exception.

- This means that you can create a method that only throws an exception but does not necessarily deal with it.

- Besides writing code that throws an exception, to indicate to the compiler that a method is throwing an exception, after it closing parenthesis, type the **throws** keyword followed by the class name of the type of exception that will be thrown

```
Return_type method_name() throws Exception{}
```

Lecture0814

- Check lab manual

Lecture0815

- Check lab manual

- In reality, even if a method does not explicitly throw an exception, you can still include a **throws Exception** expression to it if you want, in anticipation to a bad behavior.

- There is a rule you must follow.
- If a method has code that throws a general exception of type **Exception**, you must add a **throws Exception** expression to it.
- If you violate this rule, the program will not compile.
- But If a method has code that throws a specific exception of type **Exception**, (**InputMismatchException**) you don't have to add a **throws Exception** expression to it

Error

```
private static double getSide() {
        double side = 0;
        Scanner scnr = new Scanner(System.in);
        System.out.println("Square Processing");
        System.out.print("Enter Side: ");
        side = scnr.nextDouble();
        if( side < 0 )
                throw new Exception("Positive number required");
        return side;
}
```

```
private static double getSide() {
        double side = 0;
        Scanner scnr = new Scanner(System.in);
        System.out.println("Square Processing");
        System.out.print("Enter Side: ");
        side = scnr.nextDouble();
        if( side < 0 )
                throw new InputMismatchException("Positive number required");
        return side;
}
```

OK


```
private static double getSide() throws Exception {
        double side = 0;
        Scanner scnr = new Scanner(System.in);
        System.out.println("Square Processing");
        System.out.print("Enter Side: ");
        side = scnr.nextDouble();
        if( side < 0 )
                throw new Exception("Positive number required");
        return side;
}
```

OK

Lecture0816

Lecture0817

Thanks,

See you next Lecture, isA