CS202: IT Workshop Java

Exception

Ref:
1. Harvey Deitel, Paul Deitel, **Java: How to Program**, 9/e, Prentice Hall India.
2. Herb Schildt, **Java: The Complete Reference**, 8/e Tata Mcgraw Hill Education.

Discussion on MidSem questions and solutions

Solution is uploaded on course website (under Resource tab)

Do you want to **remove** negative marking in midsem (if permitted)?

A. Yes
B. No
C. Does not matter

Problems occurred at runtime

```
System.out.print( "Please enter an integer numerator: " );
int numerator = scanner.nextInt();
System.out.print( "Please enter an integer denominator: " );
int denominator = scanner.nextInt();

int result = numerator / denominator;
```

- If we enter 0 for denominator, the program **terminates** with following errors

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at cs202/exception.ExceptionDemo.main(ExceptionDemo.java:16)
```

Stack tree trace: Name of the exception, problem and the point where the problem occurred.

Class name and line number

Problems occurred at runtime

- ❑ If we enter a string instead of an integer, the program terminates with following errors

```
$ Please enter an integer numerator: 21
$ Please enter an integer numerator: hello

Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at cs202/exception.ExceptionDemo.main(ExceptionDemo.java:15)
```

- ❑ **Do we want such abrupt termination of programs?**
- ❑ If **no**, we are to **handle** this properly (execution should continue or terminate gracefully with control)
- ❑ Aim is to design **robust** and **fault-tolerant** programs

Exception handling example in Java

- **Try**: Enclose code segment that may raise an exception

```
Try {
        System.out.print( "Please enter an integer numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Please enter an integer denominator: " );
        int denominator = scanner.nextInt();

        int result = numerator / denominator;
}
```

- **Catch**: action to be taken in case an exception occurs

```
catch ( <Exception type object>) {
        // action
}
```

Code (ExceptionDemo.java): Screen share

What causes an exception?

- Any unwanted situation that occurs at runtime
- User enters invalid data
- Accessing an array element outside its range
- A file that needs to be opened cannot be found
- A network connection has been lost in the middle of communications, JVM has run out of memory, etc.

- Java categories the exceptions into three main types:
  - **Checked / Compile time:** notifies at compilation time; must handle if we write exception causing codes!
  - **Unchecked / Runtime**: programming bug, invalid input; not compulsory to handle!
  - **Errors:** beyond the control of the user or programmer

- Some common checked exceptions are:
  FileNotFoundException, SQLException, etc.

Some unchecked exception types in Java

Exception Type	Meaning
ArithmeticException	Errors due to arithmetic operation such as divide by zero
ArrayIndexOutOfBoundsException	Array index beyond range
NegativeArraySizeException	Array creation with negative size
NullPointerException	Invalid use of null reference
NumberFormatException	Invalid conversion of a String to numeric format
UnsuportedOperationsException	Applying unsupported operations to objects

- Every checked and unchecked exception type is a subclass (direct or indirect) of class java.lang.Exception
- It supports various methods
  (e.g. getMessage(), printStackTrace() etc.)

Exception handling in Java

- Java handles exception using five keywords: **try, catch, finally, throw, throws**
- **Try** must be followed by at least a catch or finally
- **Catch** includes different types of exceptions
- Once exception occurs inside try block, **control comes to the corresponding catch block** (first catch with matching exception type) and remaining catch blocks are skipped
- **finally** block will execute whether or not an exception is thrown in the corresponding try block
  - It is placed after the last catch
  - If no catch is there, it is written after try

Catch execution rule in Java

- If a catch handler is written to catch a superclass-type exception, it includes all subclass types of that superclass
- After executing the catch block, this program’s flow control proceeds to the **first** statement after the **last catch** block
- Thus a **subclass-exception-catch** should be written **before** **superclass-exception-catch** if we want to do anything specific for subclass
- But placing a superclass type exception helps to catch all exceptions!

Catch execution rule in Java

```
catch ( ArithmeticException ae ) {
        System.out.println(“subclass exception”);
}
catch ( Exception  e ) {
        System.out.println(“superclass exception”);
}
```

Subclass exception should be written **first**

```
catch ( Exception  e ) {
        System.out.println(“superclass exception”);
}
catch ( ArithmeticException ae ) {
        System.out.println(“subclass exception”);
}
```

This is **NOT** allowed

Code (ExceptionDemo.java): Screen share

Throw in java

- ❑ Java runtime system automatically throws known exceptions and those can be caught using **catch**
- ❑ If we want to manually throw an exception, we can do so using keyword **throw**

```
void checkAge(int age) {

        if(age<18)
          throw new AgeLessException ("Not Eligible for voting");
        else
          System.out.println("Eligible for voting");
        }
```

- ❑ We can also create **our own custom exception** and use it
- ❑ We can throw only **one** exception from a method

Code (ThrowDemo.java): Screen share

Throws in java

- If a method may cause exception but we do not handle it inside the method, we should declare using throws

```
void f() throws ArithmeticException { . . . }
```

- Caller method of this method (*f*()) need to handle this exception

```
try {
            f();
}
catch ( Exception  e ) {
            System.out.println("Exception in called method");
}
```

Code (ThrowsDemo.java): Screen share