Exception Handling and Debugging

What are Exceptions

- There are three sources that can lead to exceptions:
  - The End User
    - Garbage-in, garbage-out
  - The Programmer
    - Misunderstanding requirements and/or contract
  - The Environment
    - The VM, OS, hardware, etc.

Exception Example

Given two integers, write a program to compute and output their quotient.

```
System.out.println("Enter the first integer:");
int a = input.nextInt();
System.out.println("Enter the second:");
int b = input.nextInt();

int c = a / b;
System.out.println("Their quotient is: " + c);
```

Exception Example (cont.)

```
Enter the first integer:
8
Enter the second:
0
Exception in thread "main"
java.lang.ArithmeticException: / by zero
    at Quotient.main(Quotient.java:16)
```

In this case:

- The error source is the end user.
- The incorrect operation is invalid
- The exception was not caught

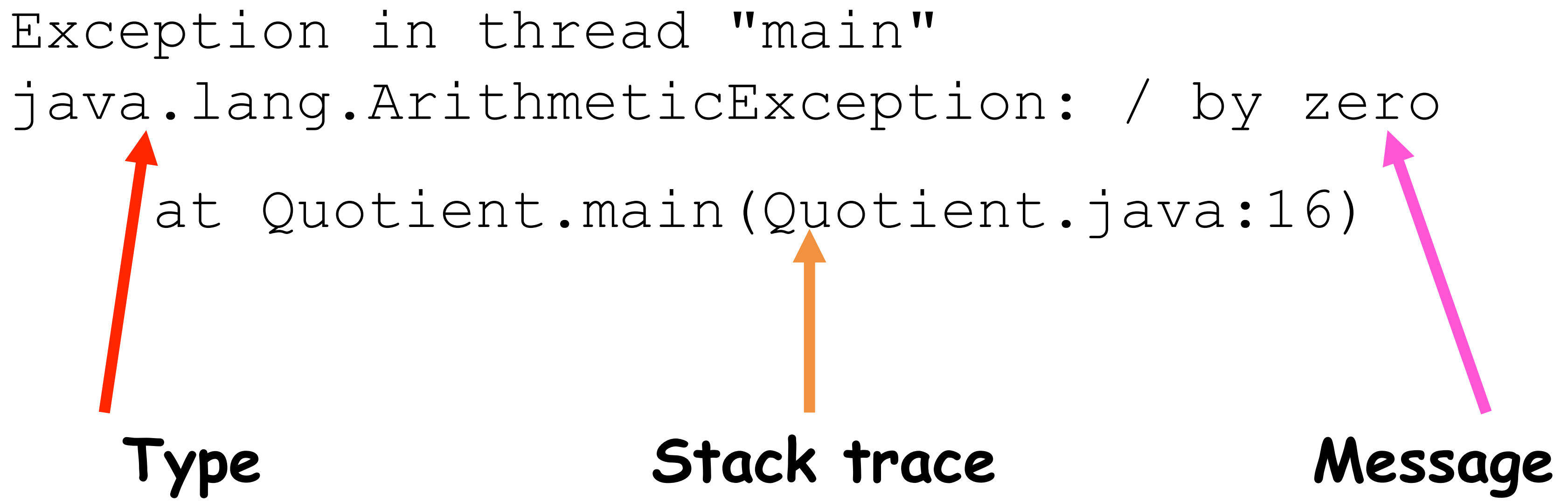
Anatomy of an Error Message

```
Enter the first integer:
8

Enter the second:
0

Exception in thread "main"
java.lang.ArithmeticException: / by zero

    at Quotient.main(Quotient.java:16)
```

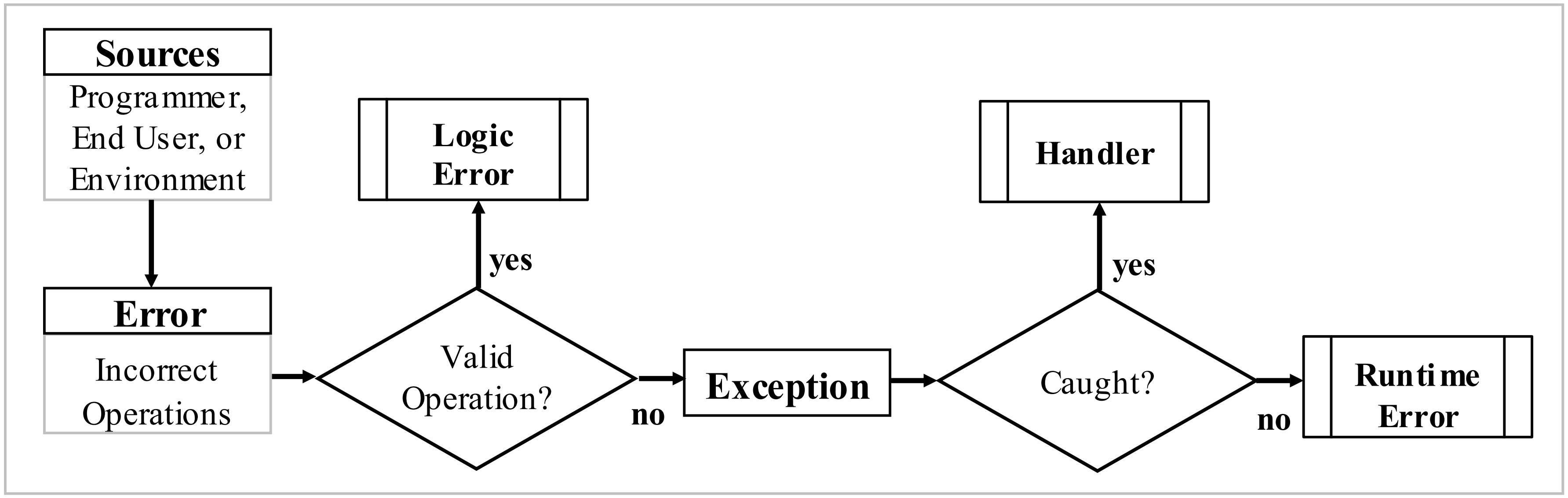


Types of Exceptions

- Unchecked
  - Does not need to be handled by code
  - Derives from RuntimeException class:
    - ArithmeticException
    - ArrayIndexOutOfBoundsException
    - …
- Checked
  - Needs to be handled by code (e.g., “throws” clause)
  - Derives from Exception class:
    - FileNotFoundException
    - IOException
    - …

Exception Handling in General

- **An error source can lead to an incorrect operation**
- **An incorrect operation may be valid or invalid**
- **An invalid operation throws an exception**
- **An exception becomes a runtime error unless caught**



Try-Catch Block

```
try
{   ...
    code fragment
    ...
}
catch (SomeType e)
{   ...
    exception handler
    ...
}
program continues
```

Examples

- Example 1:
  - Prompt for a filename and output a message if the file is not found
- Example 2:
  - Prompt for a filename
  - Loop until the user provides a valid filename

Example 1

```
Scanner input = new Scanner(System.in);
Scanner fileInput;
try
{
   System.out.print("Enter filename: ");
   fileInput = new Scanner(new File(input.nextLine()));
   System.out.println("File opened.");
}
catch (FileNotFoundException fnfe)
{
   System.out.println("File not found!");
}
```

Example 2

```
for (boolean repeat = true; repeat; )
{
   try
   {
      System.out.print("Enter filename: ");
      fileInput = new Scanner(new File(input.nextLine()));
      repeat = false;
   }
   catch (FileNotFoundException fnfe)
   {
      System.out.println("File not found! Try again.");
   }
}
System.out.println("File opened.");
```

Try-Catch with Multiple Exceptions

```
try
{   ...
}
catch (Type-1 e)
{   ...
}
catch (Type-2 e)
{   ...
}
...
catch (Type-n e)
{   ...
}
program continues
```

Why not just catch Exception?

- Each different exception class represents a different (invalid) situation
- Catching a specific exception allows you to handle that particular circumstance
- Example:
  - When you catch the FileNotFoundException, you know that the file you attempted to access does not exist

Debugging with Eclipse

- Demonstrated in lecture