Java Exception Handling
try, throw, and catch

Gerry Howser¹

¹Kalamazoo College, Kalamazoo, Michigan USA

COMP 210, Spring 2017
Outline

1. Introduction
   - Exceptions in Java

2. Creating an Exception
   - Extending the class Exception

3. throwing a Custom Exception

4. try–catch Blocks
   - try
   - catch
   - More on Catching Exceptions
Exceptions in Java

Exceptions (a.k.a oops!)

Background

- Exceptions are all subclasses of the class `throwable`.
- Error subclass is for "hard" errors such as catastrophic error.
- Run-time exceptions are for run-time errors that may (or may not) be fixed on the fly.
Exceptions (a.k.a oops!)

Background

- Exceptions are all subclasses of the class `throwable`.
- Error subclass is for “hard” errors such as catastrophic error.
- Run-time exceptions are for run-time errors that may (or may not) be fixed on the fly.
Exceptions in Java

Exceptions (a.k.a oops!)

Background

- Exceptions are all subclasses of the class `throwable`.
- Error subclass is for “hard” errors such as catastrophic error.
- Run-time exceptions are for run-time errors that may (or may not) be fixed on the fly.
The Class `throwable`
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. **Don’t.**
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. **Don’t.**
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. Don’t.
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. Don’t.
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Exceptions in Java

Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. **Don’t.**
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. **Don’t.**
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. *Don’t.*
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. *Don’t.*
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. Don’t.
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. Don’t.
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. **Don’t.**
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. **Don’t.**
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Three Types of Exceptions

- **error subclasses**
  - Technically, you can throw errors. **Don’t.**
  - These are catastrophic failures and usually can’t be fixed by the user

- **runtime error subclasses**
  - Technically, you can throw these as well. **Don’t.**
  - These are usually JAVA errors

- **exception errors**
  - Java has many exceptions you might want to throw/catch
  - This is the best place to create your own exceptions
Exceptions in Java

What to throw

- You should never throw an error, only an exception
- What exceptions a method can throw must be documented
- What exceptions a method can throw are part of the method signature
- You must instantiate an exception when you throw it
- A good place to start looking:

  [ furnished link: https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html ]
Exceptions in Java

What to throw

- You should never throw an error, only an exception
- What exceptions a method can throw must be documented
- What exceptions a method can throw are part of the method signature
- You must instantiate an exception when you throw it
- A good place to start looking:

  https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html
Exceptions in Java

**What to throw**

- You should never **throw an error**, only an **exception**.
- What exceptions a method can throw **must** be documented.
- What exceptions a method can throw are part of the method signature.
- You must instantiate an exception when you throw it.
- A good place to start looking:
  
  https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html
What to throw

- You should never throw an error, only an exception
- What exceptions a method can throw must be documented
- What exceptions a method can throw are part of the method signature
- You must instantiate an exception when you throw it
- A good place to start looking:
  https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html
What to throw

- You should never throw an error, only an exception
- What exceptions a method can throw must be documented
- What exceptions a method can throw are part of the method signature
- You must instantiate an exception when you throw it
- A good place to start looking:

  https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html
Outline

1. Introduction
   - Exceptions in Java

2. Creating an Exception
   - Extending the class Exception

3. throwing a Custom Exception

4. try–catch Blocks
   - try
   - catch
   - More on Catching Exceptions
Creating an Exception to throw

You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else’s.

- Do you need an exception type that isn’t represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else’s exceptions, will users have access to those exceptions?
- A similar question is, should your package be independent and self-contained?
Creating an Exception to throw

You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else’s.

- Do you need an exception type that isn’t represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else’s exceptions, will users have access to those exceptions?
- A similar question is, should your package be independent and self-contained?
Creating an Exception to throw

You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else’s.

- Do you need an exception type that isn’t represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else’s exceptions, will users have access to those exceptions?
- A similar question is, should your package be independent and self-contained?
Creating an Exception to throw

You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else’s.

- Do you need an exception type that isn’t represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else’s exceptions, will users have access to those exceptions?
- A similar question is, should your package be independent and self-contained?
Creating an Exception to throw

You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else’s.

- Do you need an exception type that isn’t represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else’s exceptions, will users have access to those exceptions?
- A similar question is, should your package be independent and self-contained?
Guidelines for Your Exceptions

- For readable code, it’s good practice to append the string Exception to the names of all classes that inherit (directly or indirectly) from the Exception class.

- If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.
Guidelines for Your Exceptions

- For readable code, it’s good practice to append the string `Exception` to the names of all classes that inherit (directly or indirectly) from the `Exception` class.

- If a client can reasonably be expected to recover from an exception, make it a checked exception. If a client cannot do anything to recover from the exception, make it an unchecked exception.
Suppose you wish to create an exception if a user tries to pop() an empty stack.

- Change the method signature: `element method pop() throws StackEmptyException`
- Create a class file named `StackEmptyException`
- Usually, you can simply create a new class and then use it:
  
  ```java
  public class StackEmptyException extends Exception
  ```
Suppose you wish to create an exception if a user tries to \texttt{pop()} an empty stack.

- **Change the method signature:** element method \texttt{pop()} throws \texttt{StackEmptyException}
- **Create a class file named** \texttt{StackEmptyException}

Usually, you can simply create a new class and then use it:

- **public class** \texttt{StackEmptyException} extends \texttt{Exception}
Suppose you wish to create an exception if a user tries to pop() an empty stack.

- Change the method signature: `element method pop() throws StackEmptyException`
- Create a class file named `StackEmptyException`
- Usually, you can simply create a new class and then use it:
  ```java
  public class StackEmptyException extends Exception
  ```
Suppose you wish to create an exception if a user tries to \texttt{pop()} an empty stack.

- Change the method signature: \texttt{element method pop()} \texttt{throws StackEmptyException}
- Create a class file named \texttt{StackEmptyException}
- Usually, you can simply create a new class and then use it:
  - \texttt{public class StackEmptyException extends Exception}
Introduction

Creating an Exception

throwing a Custom Exception

try-catch Blocks

Outline

1. Introduction
   - Exceptions in Java

2. Creating an Exception
   - Extending the class Exception

3. throwing a Custom Exception

4. try-catch Blocks
   - try
   - catch
   - More on Catching Exceptions
pop() with Error

```java
public element pop() throws StackEmptyException {
if (this.isEmpty())
{
    throw new StackEmptyException;
}
}
```
Outline

1. Introduction
   - Exceptions in Java

2. Creating an Exception
   - Extending the class Exception

3. throwing a Custom Exception

4. try–catch Blocks
   - try
   - catch
   - More on Catching Exceptions
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
  - If the user can fix it, catch it
- Unchecked Exceptions (exceptions the user can’t fix)
  - No one can help, don’t catch it.
- Exceptions that are not caught abort execution.
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
  - If the user can fix it, catch it
- Unchecked Exceptions (exceptions the user can’t fix)
  - No one can help, don’t catch it.
- Exceptions that are not caught abort execution.
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
- If the user can fix it, catch it
- Unchecked Exceptions (exceptions the user can’t fix)
- No one can help, don’t catch it.
- Exceptions that are not caught abort execution.
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
- If the user can fix it, *catch* it
- Unchecked Exceptions (exceptions the user can’t fix)
- No one can help, *don’t catch* it.
- Exceptions that are not caught abort execution.
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
- If the user can fix it, catch it
- Unchecked Exceptions (exceptions the user can’t fix)
- No one can help, don’t catch it.
- Exceptions that are not caught abort execution.
Using Exceptions

- There are two types of exceptions (for now)
- Checked Exceptions (exceptions the user can fix)
- If the user can fix it, catch it
- Unchecked Exceptions (exceptions the user can’t fix)
- No one can help, don’t catch it.
- Exceptions that are not caught abort execution.
If you can fix the error (with or without user “help”) you should surround statements that can cause the error with a `try` block.

```java
try {
    target = myStack.pop();
}
```
Once an exception has been thrown, you have three choices

- Ignore the error and allow the program to terminate
- `catch` the error and fix it
- `catch` the error and have the user fix it (enter a different file name???)
try-catch Blocks

```java
.....
int x = 10;
int y = 10;
try{
    int num = x/y;
    System.out.println("next-statement: Inside try block");
}catch(Exception ex)
{
    System.out.println("Exception");
}
System.out.println("next-statement: Outside of try-catch");
...
```

Output:

```
next-statement: Inside try block
next-statement: Outside of try-catch
```

Figure: From: http://beginnersbook.com/2013/05/flow-in-try-catch-finally/
Multiple `catch` Blocks

If a `try` block can throw more than one exception that you can fix, you will need a `catch` block for each one. If you wish some common code to be executed after exceptions occur, you can place that in a `finally` block.
More on Catching Exceptions

Example of try-catch-finally Blocks

class TestExceptions {
    static void myMethod(int testnum) throws Exception {
        System.out.println("start - myMethod");
        if (testnum == 12)
            throw new Exception();
        System.out.println("end - myMethod");
        return;
    }
    public static void main(String[] args) {
        int testnum = 12;
        try {
            System.out.println("try - first statement");
            myMethod(testnum);
            System.out.println("try - last statement");
        } catch (Exception ex) {
            System.out.println("An Exception");
        }
        finally {
            System.out.println("finally");
        }
    }
    System.out.println("Out of try/catch/finally - statement");
}
More on Catching Exceptions

Output from Example

Output:

```
try - first statement
start - myMethod
An Exception
finally
Out of try/catch/finally - statement
```

Figure: Output from the Example