Lecture 5

Exceptions

Some slides were borrowed from Josh Hug and Adam Jundt

Reminders

1) Mic
2) PA01 feedback (part of the grade), on Canvas
    a) For each PA
    b) After the submission period is over.

Exceptions

Exceptions

- An exception is abnormal condition in which the normal execution of a code gets interrupted.

- Open a file
- Divide by zero
- Access invalid cell in array
- ...

Exceptions

- **2 types**
  - **Checked exceptions**: you have to catch those, otherwise the program will not compile: (java.lang.Exceptions)
    - ClassNotFound
    - IOExceptions

Exceptions

- **2 types**
  - **Checked exceptions**: you have to catch those, otherwise the program will not compile.
  - **Runtime (unchecked) exceptions**: problems during runtime: java.lang.RunTimeException
    - ArithmeticException
    - IndexOutOfBounds
    - IllegalArgumentException
    - ...

Handling Errors

```
Exception in thread "main"
java.lang.IllegalArgumentException: Please make sure that
the input is not empty
    at ThrowExample.example(ThrowExample.java:6)
    at Driver.main(Driver.java:9)
```

Sometimes things go wrong, e.g.

- You try to use 383,124 gigabytes of memory.
- You try to call a method using a reference variable that is equal to null.
- You try to access index -1 of an array.

The Java approach to handling these exceptional events is to ***`throw`*** an ***exception***.

- Disrupts normal flow of the program.
- You saw that these exceptions just cause the program to crash, printing out a helpful (?) message for the user.

Exceptions: May be Explicitly or Implicitly Thrown

Java itself can throw exceptions *implicitly*, e.g.

```
String t = "Marina";
int n = Integer.parseInt(t);
```

```
Exception in thread "main" java.lang.NumberFormatException:
For input string: "Marina"
    at java.lang.NumberFormatException.forInputString(
    NumberFormatException.java:65)
```

Exceptions: May be E

Java itself can th

```
String t = "Marin
int n = Integer.parseInt(t);
```

```
Exception in thread "main" java.lang.NumberFormatException:
For input string: "Marina"
    at java.lang.NumberFormatException.forInputString(
    NumberFormatException.java:65)
```

Java code can also throw exceptions explicitly using ***throw*** keyword:

```
public static void main(String[] args) {
    System.out.println("ayyy lmao");
    throw new RuntimeException("For no reason.");
}
```

Creates new object of type RuntimeException!

```
$ java Alien
ayyy lmao
Exception in thread "main"
java.lang.RuntimeException: For no
reason.
    at Alien.main(Alien.java:4)
```

Problem

```
class Exceptions {

    static int  num = 10;
    public static void main(..) {
     double result = divideByNumber(0);
    }

    public static double divideByNumber (int d)    {
        return num/d;
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Exceptions.divideByNumber(Exceptions.java:10)
    at Exceptions.main(Exceptions.java:6)
```

TRY... CATCH

```
try {
   // code
} catch(ExceptionType e) {
   // code
}
```

Solution

```
class Exceptions {

    static int  nu
    public static
     double result
    }

    public static
        return nu
    }
}
```

```
public static double divideByNumber (int d) {

    try {
        return num/d;
    }
    catch(ArithmeticException e) {
        System.out.println ("Oh-Oh. 0 will be return as an error");
        return 0;

    }
}
```

```
Oh-Oh. 0 will be printed as an error(
```

Solution

```java
class Exceptions {

      static int  num = 10;
      public static void main(..) {
      double result = divideByNumber(0);
      }

      public static double divideByNumber (int d) {
            return num/d;
      }
}
```

```java
public static double divideByNumber (int d) {

    try {
        return num/d;
    }
    catch(ArithmeticException e) {
        System.out.println (e.getMessage());
        System.out.println (e.toString());
        return 0;

    }
}
```

```
/ by zero
java.lang.ArithmeticException: / by zero
[Finished in 1.1s]
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Exceptions.divideByNumber(Exceptions.java:10)
    at Exceptions.main(Exceptions.java:6)
```

TRY... CATCH AGAIN

```
try {
              // code
}
catch(IndexOutOfBounds e) {
            // code
 }

catch(ArithmeticExpection e) {
            // code
 }
```

Try _catch and Declaring exceptions

Declaring an exception. User's problem (demo)

- Methods can cause an exception, and instead of fixing it, just report to whoever called it

```
class Exceptions {

    static int [] nums = {1, 2, 3};
    public static void main(String [ ]args) {

        int [] result = doubleArray();
    }

    public static int [] doubleArray() throws IndexOutOfBoundsException {

    int b  [] = new int[nums.length];
    for(int i = 0; i < nums.length; i++) {

        b[i] = nums[i] * 2;
    }
    return b;
  }

}
```

Method pop from Java doc

https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html#pop-

There are two ways to handle exceptions:

1. Use `try/catch` block
2. Use `throws` clause

Throw and checked exceptions, demo

Throw: Explicitly throw an exception

```java
class Test {
    static void testing(int i) {
        if (i < 20){
            throw new IllegalArgumentException("Too Young");
        }
        else if (i>50) {
            throw new IllegalArgumentException("Too Old");
            }
        else {
                System.out.print("Welcome to my party!");
            }
    }
}
```

```java
class Example {
    public static void main(String[] args) {
        Test.testing(16);
        Test.testing(25);
        Test.testing(75);
    }
}
```

A:
IllegalArgumentException:
Too Young

B:
IllegalArgumentException:
Too Young

Welcome to my party!

C:
IllegalArgumentException:
Too Young

Welcome to my party!

IllegalArgumentException:
Too Old

D: Something else

Checked Exception (code will not compile)

Slides were borrowed from Josh Hug

“Must be Caught or Declared to be Thrown”

Occasionally, you’ll find that your code won’t even compile, for the mysterious reason that an exception *“must be caught or declared to be thrown”.*

- The basic idea: Some exceptions are considered so disgusting by the compiler that you MUST handle them somehow.
- We call these “checked” exceptions.

```
public static void main(String[] args) {
    Eagle.gulgate();
}
```

```
$ javac What.java
What.java:2: error: unreported exception IOException; must be caught or
declared to be thrown
        Eagle.gulgate();
        ^
```

Checked Exceptions

Examples so far have been ***unchecked*** exceptions. There are also ***checked*** exceptions:

To be defined soon...

- Compiler requires that these be “caught” or “specified”.
  - Goal: Disallow compilation to prevent avoidable program crashes.
- Example:

```
public class Eagle {
    public static void gulgate() {
        if (today == “Wed”) {
            throw new IOException("hi"); }
    }
}
```

```
$ javac Eagle
Eagle.java:4: error: unreported exception IOException; must be caught
or declared to be thrown
        throw new IOException("hi"); }
        ^
```

Unchecked Exceptions

By contrast unchecked exceptions have no such restrictions.

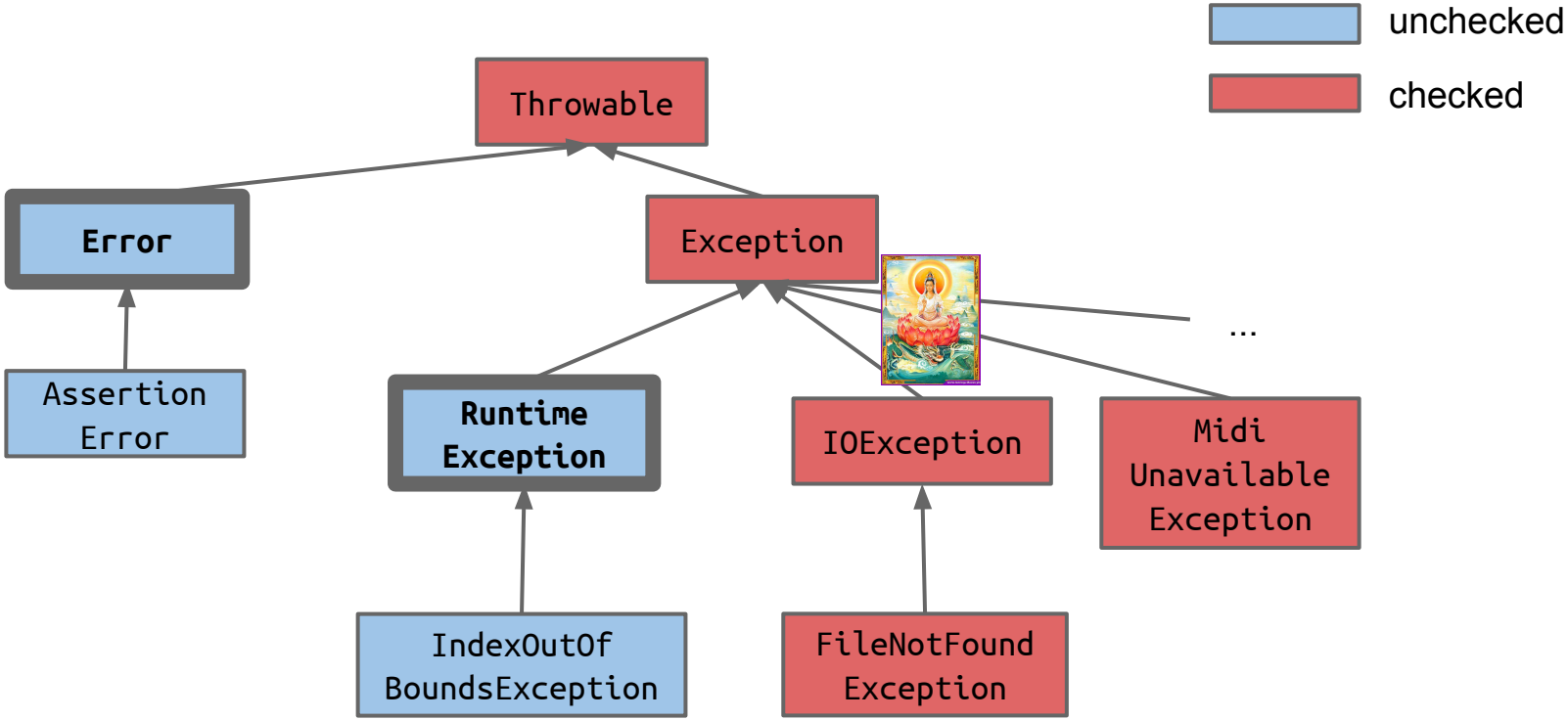
- Code below will compile just fine (but will crash at runtime).

```
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        if (today == “Wed”) {
            throw new RuntimeException("as a joke"); }
    }
}
```

```
$ javac UncheckedExceptionDemo.java
$ java UncheckedExceptionDemo
Exception in thread "main" java.lang.RuntimeException: as a joke.
    at UncheckedExceptionDemo.main(UncheckedExceptionDemo.java:3)
```

Checked vs. Unchecked Exceptions

Any subclass of **RuntimeException** or **Error** is *unchecked*, all other Throwables are *checked*.



Checked vs. Unchecked

- Compiles fine, because the possibility of unchecked exceptions is allowed:

```
public class UncheckedExceptionDemo {
    public static void main(String[] args) {
        if (today == “Wed”) {
            throw new RuntimeException("as a joke"); }
    }
}
```

Java considers this an “unchecked” exception.

- Won’t compile, because there exists possibility of checked excep

Why didn’t you catch or specify??

```
public class Eagle {
    public static void gulgate() {
        if (today == “Wed”) {
            throw new IOException("hi"); }
    }
}
```

Java considers this a “checked” exception.

Checking Exceptions

Compiler requires that all checked exceptions be **caught** or **specified**.

Two ways to satisfy compiler:

- **Catch**: Use a catch block after potential exception.

```
public static void gulgate() {
    try {
         if (today == “Wed”) {
           throw new IOException("hi"); }
    } catch (Exception e) {   #is ok
        System.out.println("psych!");
    }
}
```

- **Specify** method as dangerous with ***throws*** keyword.

Checking Exceptions

Compiler requires that all checked exceptions be **caught** or **specified**.

Two ways to satisfy compiler:

- **Catch**: Use a catch block after potential exception.
- **Specify** method as dangerous with ***throws*** keyword.
  - Defers to someone else to handle exception.

```
public static void gulgate() throws IOException {
   ... throw new IOException("hi"); ...
}
```

Checking Exceptions

If a method uses a ‘dangerous’ method (i.e. might throw a checked exception), it becomes dangerous itself.

```
public static void gulgate() throws IOException {
    ... throw new IOException("hi"); ...
}
```

```
public static void main(String[] args) {
    Eagle.gulgate();
}
```

```
$ javac What.java
What.java:2: error: unreported exception IOException;
must be caught or declared to be thrown
        Eagle.gulgate();
        ^
```

How do we fix this?

Catch or specify!

Checking Exceptions

Two ways to satisfy compiler: *Catch* or *specify* exception.

```
public static void gulgate() throws IOException {
    ... throw new IOException("hi"); ...
}
```

```
public static void main(String[] args) {
  try {
    Eagle.gulgate();
  } catch(IOException e) {
    System.out.println("Averted!");
  }
}
```

Catch an Exception:
Keeps it from getting out.

Use when you can handle the problem.

```
public static void main(String[] args)
    throws IOException {
      Eagle.gulgate();
}
```

Specify that you might throw an exception.

Use when someone else should handle.

Throws vs throw

```
public int pop() throws EmptyStackException {
    int obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

Not checked, so

```
public int pop() throws EmptyStackException {
    int obj;

    if (size == 0) {
        throw new EmptyStackException();
    }

    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```

https://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/

Why Exceptions?

Allows you to keep error handling code separate from ‘real’ code.

- Consider pseudocode that reads a file:

```
func readFile: {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

what if the file doesn’t exist?

what if there’s not enough memory?

what happens if reading fails?

Error Handling Code (Naive)

One naive approach to the right.

- Clearly a bad idea.

```
func readFile: {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

```
func readFile: {
    open the file;
    if (theFileIsOpen) {
        determine its size;
        if (gotTheFileLength) {
            allocate that much memory;
         } else {
             return error("fileLengthError");
         }
         if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
               return error("readError");
            }
           ...
        } else {
          return error("memoryError");
        }
     } else {
        return error("fileOpenError")
     }
}
```

With Exceptions

```
func readFile: {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

```
func readFile: {
    open the file;
    if (theFileIsOpen) {
        determine its size;
        if (gotTheFileLength) {
            allocate that much memory;
        } else {
            return error("fileLengthError");
        }
        if (gotEnoughMemory) {
            read the file into memory;
            if (readFailed) {
                return error("readError");
            }
            ...
        } else {
            return error("memoryError");
        }
    } else {
        return error("fileOpenError")
    }
}
```