Java Exceptions

Colorado State University
Department of Computer Science

Slides Originally Created by Albert Lionelle (Albert.Lionelle@colostate.edu),
updated by Marcia Moraes (marcia.moraes@colostate.edu)

https://www.amazon.com/Inspirational-Motivational-Paintings-Educational-Classroom/dp/B0B5THMZ8F

Announcements

TODO Reminders:

Readings are due **before** lecture

- Reading 17 (zybooks) – you should have already done that ☺
- Lab 11
- Reading 18 (zyBooks) – you should have already done that ☺
- Lab 12
- Reading 19 (zybooks)

- RPA 9

Keep practicing your RPAs in a spaced and mixed manner ☺

Help Desk

Day	Time : Room
Monday	3 PM - 5 PM : CSB 120
Tuesday	6 PM - 8 PM : Teams
Wednesday	3 PM - 5 PM : CSB 120
Thursday	6 PM - 8 PM : Teams
Friday	3 PM - 5 PM : CSB 120
Saturday	12 PM - 4 PM : Teams
Sunday	12 PM - 4 PM : Teams

Z8F

Recall Activity

- What are exceptions in Java? Explain providing an example.

https://rollbar.com/blog/java-exceptions-hierarchy-explained/

What are Exceptions?

- Classes / Objects!
  - They contain information about the error that is happening

- What about try/catch and throws?
  - Those are commands that use those objects!

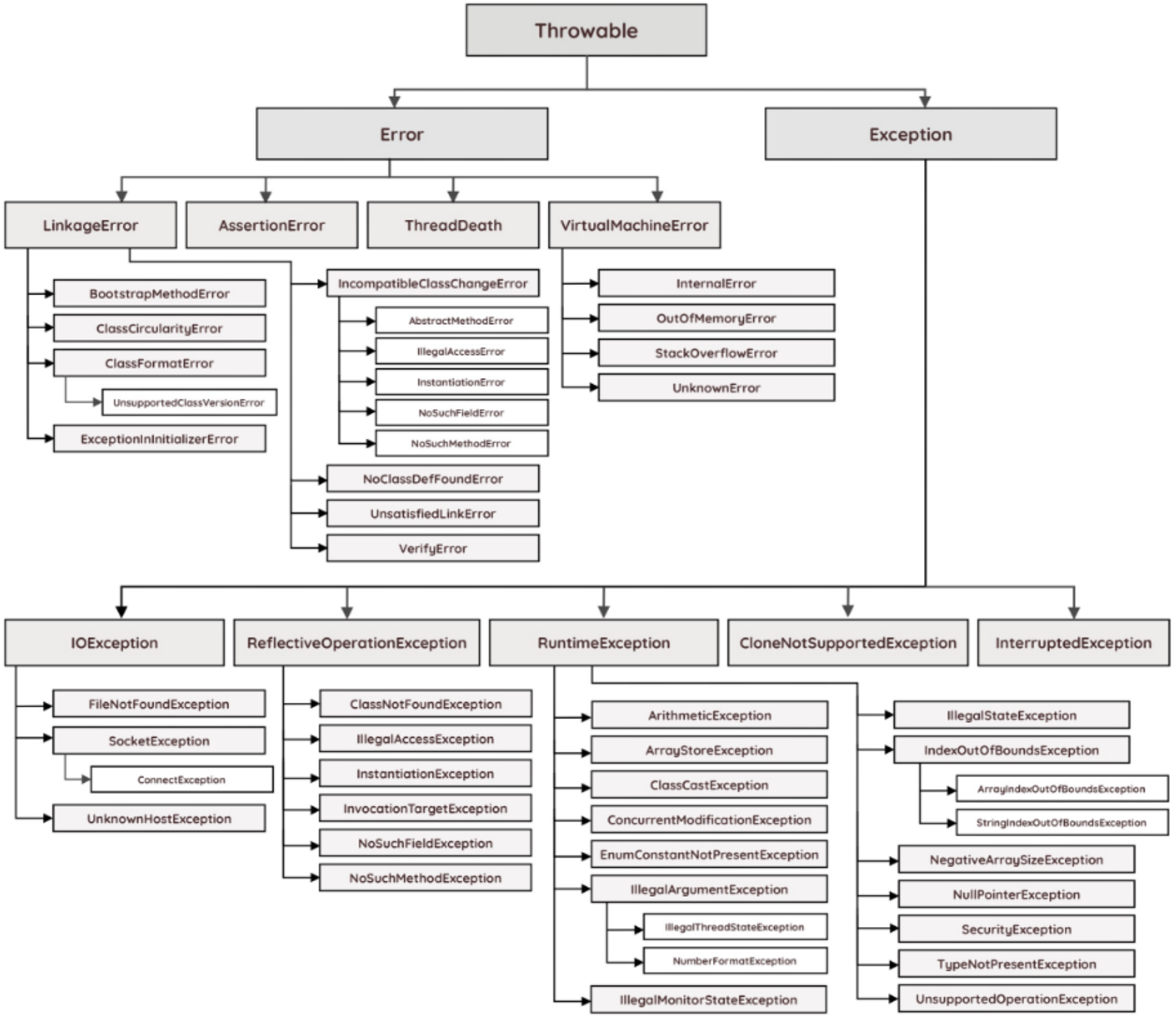
try/catch

- `try{}` - Says "try this block of code"
- `catch(Exception x)` {} - run this block of code if there is an error
- `finally {}` - always run this block of code error or not (often can be omitted, won't be used much in this class)

https://rollbar.com/blog/java-exceptions-hierarchy-explained/

Java Exception Hierarchy

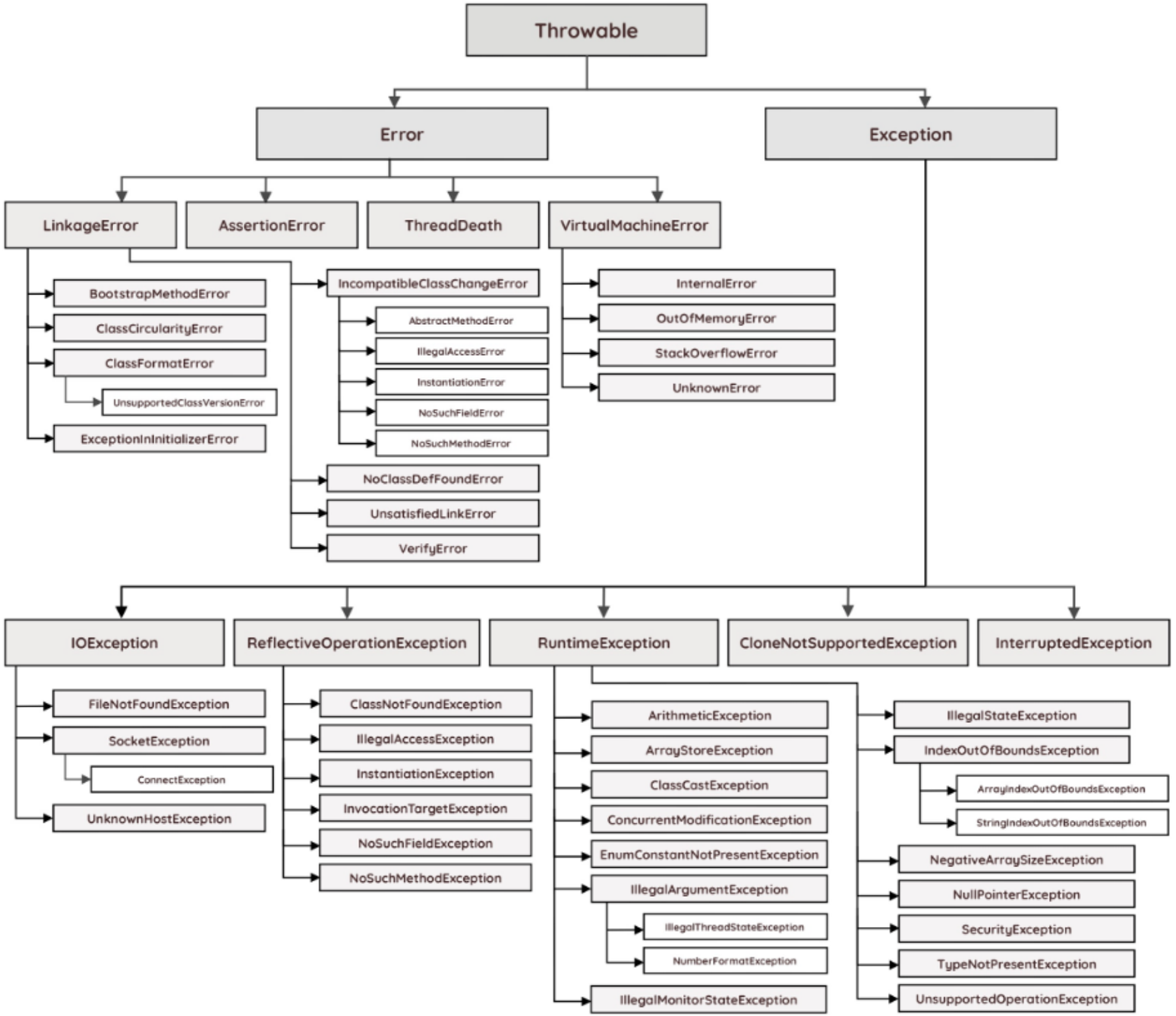
- Error class is used to indicate a more serious problem in the architecture and should not be handled in the application code.
- Exception class is used for exception conditions that the application may need to handle.
- Exceptions are further subdivided into checked (compile-time) and unchecked (run-time) exceptions.



https://rollbar.com/blog/java-exceptions-hierarchy-explained/

Java Exception Hierarchy

- Exceptions that can occur at **compile-time** are called **checked exceptions** since they need to be explicitly checked and handled in code.
  - All classes with the exception of Error and RuntimeException are checked
- Unchecked exceptions can be thrown "at any time" (i.e. run-time). Therefore, methods don't have to explicitly catch or throw unchecked exceptions.
  - RuntimeException



https://rollbar.com/blog/java-exceptions-hierarchy-explained/

Controlling Exceptions

- We can't control every possible error situation
- For example:
  - What happens if the file is not there?
  - What if you don't have permission to read it?
  - Not just files
    - What about network connections?
    - What if printers aren't there?
- Exception handling
  - try – catch

```
try {
    fileScanner = new Scanner(new File(file));


}catch (IOException io) {
    io.printStackTrace();
}
```

Try - Catch

- **try**
  - try a block of code.
  - If it runs properly, great!
- **catch**
  - an exception happened!
  - run the catch block of code
- **throws**
  - allows you to "throw" the exception
  - requires someone else to handle it
- **Exception**
  - an object / class we use for errors!
  - You can write you own
    - or use built in cases
  - Checked (compile time) or Unchecked (run time)
    - Checked requires try/catch

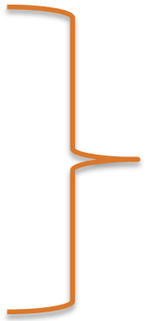
- IOException - Input / output exception - **checked**
- FileNotFoundException - Subset of IOException, but allows you to specify exactly that it is file not found **- checked**
- ArrayIndexOutOfBoundsException - you don't usually try/catch this, but you can **- unchecked**
- NullPointerException - you don't usually try/catch this **- unchecked**
- NumberFormatException - Shows up when you take a string that doesn't look like a number, and try to make it a number **- unchecked**

```java
import java.util.Scanner;

public class BMIExceptHandling {
   public static int getWeight(Scanner scnr) throws Exception {
      int weightParam;      // User defined weight (lbs)
      // Get user data
      System.out.print("Enter weight (in pounds): ");
      weightParam = scnr.nextInt();
      // Error checking, non-negative weight
      if (weightParam < 0) {
         throw new Exception("Invalid weight.");
      }
      return weightParam;
   }

   public static int getHeight(Scanner scnr) throws Exception {
      int heightParam;      // User defined height (in)
      // Get user data
      System.out.print("Enter height (in inches): ");
      heightParam = scnr.nextInt();
      // Error checking, non-negative height
      if (heightParam < 0) {
         throw new Exception("Invalid height.");
      }
      return heightParam;
   }
```

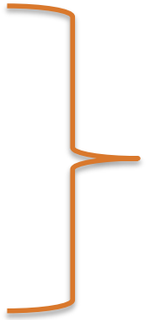
```java
public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int weightVal;   int heightVal;  float bmiCalc;
      char quitCmd;
      quitCmd = 'a';
      while (quitCmd != 'q') {
         try {
            //Get user data
            weightVal = getWeight(scnr);
            heightVal = getHeight(scnr);
            // Calculate BMI and print user health info if no input error
            // Source: http://www.cdc.gov/
            bmiCalc = ((float) weightVal /
                  (float) (heightVal * heightVal)) * 703.0f;
            System.out.println("BMI: " + bmiCalc);
            System.out.println("(CDC: 18.6-24.9 normal)");
         } catch (Exception excpt) {
            // Prints the error message passed by throw statement
            System.out.println(excpt.getMessage());
            System.out.println("Cannot compute health info");
         }
         // Prompt user to continue/quit
         System.out.print("\nEnter any key ('q' to quit): ");
         quitCmd = scnr.next().charAt(0);
      }
   }
}
```

Multiple Handlers

```
// ... means normal code
...
try {
   ...
   throw objOfExcptType1;
   ...
   throw objOfExcptType2;
   ...
   throw objOfExcptType3;
   ...
}
catch (ExcptType1 excptObj) {
   // Handle type1
}
catch (ExcptType2 excptObj) {
   // Handle type2
}
catch (Throwable thrwObj) {
   // Handle others (e.g., type3)
}
... // Execution continues here
```



Multiple exceptions can happen



Needs to handle the more specifics first than the more generic

Finally Block

```
// ... means normal code
...
try {
   ...
   // If error detected
      throw objOfExcptType;
   ...
}
catch (excptType excptObj) {
   // Handle exception, e.g., print message
}
finally {
   // Clean up resources, e.g., close file
}
...
```

Exception can happen

Handle the exception

Block of commands that executes after the program exits the corresponding try or catch blocks. It is always executed!

Example of Multiple Handlers and Finally

```
import java.io.File; import java.io.FileNotFoundException;
import java.io.IOException; import java.util.Scanner;

public class FilesException {
   private String fileName;
   public FilesException(String fileName){
      this.fileName = fileName;
   }
   public String readFile(){
      String strFromFile = "";
      Scanner scnrFile = null;
      try{
         scnrFile = new Scanner(new File(fileName));
         while(scnrFile.hasNext()){
            strFromFile += scnrFile.nextLine() + "\n";
         }
      }catch(FileNotFoundException fileExp){
         System.out.println("File not found!");
      }catch(IOException ioExp){
         System.out.println("Something wrong with file!");
      }finally {
         scnrFile.close();
      }
      return strFromFile;
   }
}
```

First handler

Second handler

Finally – always executed

Advanced: Creating your Own Exception

- You can create and throw your own exceptions (often called "raise" in other languages)
- In java, you have to **extend** the *Exception* class to do that
  - Ensures certain methods are implements for try/catch/throw/throws

- Won't use much in this class, but worth knowing
- Especially useful if you are developing an SDK (Software Development Kit)/API (Application Programming Interface)

Advanced: Creating your Own Exception

```
public class MyCoolException extends Exception {

   public MyCoolException(String msg) {
      super(msg);
    }

    public String getMessage() {
      return "SUPER COOL: " + super.getMessage();
    }
}
```

Inheritance – MyCoolException inherits all attributes and methods from Exception

Calls the super class constructor – setting the message for the Exception

Overrides the getMessage method inherited from Exception

https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

Advanced: Creating your Own Exception

```
public class MyCoolExceptionApp {
   public static void doSomething(boolean type) throws MyCoolException,Exception {
      if(type) throw new MyCoolException("This is a personal message");
      throw new Exception("General exceptions can have messages");
   }
   public static void test(boolean type) {
      try {
         doSomething(type);
      }catch(MyCoolException ex) {
         System.err.println(ex.getMessage());
      }catch(Exception ey) {
         System.err.println(ey.getMessage());
      }
   }
   public static void main(String args[]){
      test(true);
      test(false);
   }
}
```

throws – other method needs to treat both MyCoolException and Exception

throw new – creates a MyCoolException object

throw new – creates a Exception object

Since doSomething can throw two exception, we need to handle both exceptions
Ordering of the handling – more specific first (MyCoolException), then the more general (Exception)

Group Practice

- Worksheet