CS206

I/O Methods

Files/Exceptions

Inheritance

Quizlet

- What is a "Data Structure"
  - @4 reasonable answers
  - favorite answer: 1. Something about data 2. I know nothing more 3. No. 1 May be wrong.
  - Correct Answer: A way of organizing data. The simplest data structure is an array. Others we will study: ArrayList, LinkedList, Trees …
- UNIX: Hello.java in /home/YOU/cs206. Get there, compile, run.
  - 3 reasonable answers
  - Favorite Answer: "Get me home Siri"; please compile…please? Gooo!
    - second favorite: drawing of a dinosaur (Barney?)
  - Correct Answer:
    - ```
      cd /home/YOU/cs206
      javac Hello.java
      java Hello
      ```
      ```

Quizlet (part 2)

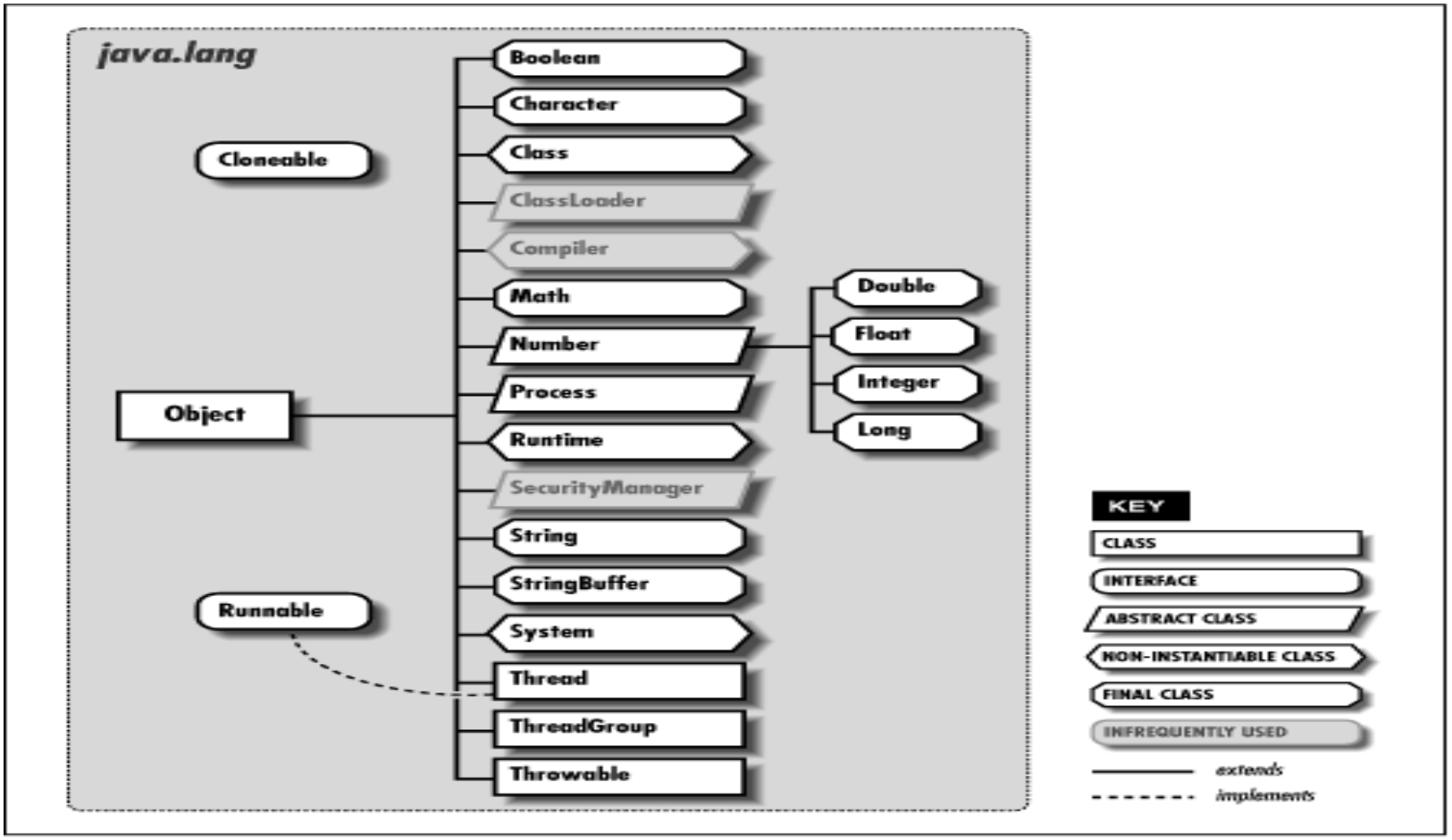
- Write a complete program that prints "Hello World" 1000 times
  - Almost everyone did something useful
  - Answer:

```
public class HelloWorld {
      public static void main(String[] args) {
               for (int i=0; i<1000; i++) {
                        System.out.println("Hello World");
               }}}
```

- What is overloading of methods?
  - 3-4 reasonable answers
  - Favorite answer: Its when methods try to list too much at the gym.
  - Answer:

```
Overloading is using the same method name and return
value with different sets of arguments.
public class Counter {
  public int timeser() {
      return 0;
  }
  public int timeser(int param) {
       return param*param;
  }}
```

Start of the Java class hierarchy



http://web.deu.edu.tr/doc/oreily/java/langref/ch10_js.htm

Java Object Methods

- **public boolean equals(Object ob)**
- **public String toString()**
- public Class getClass()
- protected Object clone()
- protected void finalize()
- public int hashCode()
- public void notify()
- public void notifyAll()
- public void wait()
- public void wait(long l)
- public void wait(long l, int ii)

Casting, Classes and Inheritance

- Suppose: SPCA shelter for only dogs and cats
- Desire: A program that tracks all animals at shelter
- Approach
  - Create 3 classes, Dog and Cat that extend (inherit from) from Animal.
  - Use single array to hold all animals
  - But deal with dogs cats separately later

```
public class Animal {}

public class Dog extends Animal {}

public class Cat extends Animal {}

public class Shelter {
    Animal[] animals = new Animal[100];
    int animalCount=0;
    public void addAnimal(Animal animal) {
        animals[animalCount++]=animal;
    }
    public Animal getAnimal(int location) {
        return animals[location];
    }
    public static void main(String[] args)
        Shelter shelter = new Shelter();
        shelter.addAnimal(new Dog());
        shelter.addAnimal(new Cat());
        Cat c = (Cat)shelter.getAnimal(1);
        System.out.println(c);
    }}
```

Strings

- Strings - "a", “abc" – double quotes
- Characters - ‘a' – single quotes
- Declaring String objects

```
String name;
String name = new String();
```

- Declaring String objects with initialization

```
String name = "Fred";
String name = new String(“Fred");
```

There are subtle differences between these two declarations.

Strings, example

```
/************************
 * @author  gtowell
 * Purpose:
 *    String sample
 * Created: August 28, 2019
 * Modified: August 29, 2019
 *           January 9, 2020
************************/
public class Stringer {
    public static void main(String[] args) {
        String geoffrey = "Geoffrey";
        String geoffrey2 = new String("Geoffrey");
        System.out.println(geoffrey);
        String geoff = geoffrey.substring(0, 5);
        System.out.println(geoff);
        String c = geoffrey.concat(geoff);
        String d = geoffrey + geoff; // + on strings does concatenation
        System.out.println("|" + geoffrey + "|" + geoff + "|" + c + "|");
        System.out.println("|" + geoffrey + "|" + geoff + "|" + d + "|");
        if (geoffrey == geoffrey2) {
            System.out.println("Same object |" + geoffrey + "||" + geoffrey2 + "|");
        }
        if (geoffrey.equals(geoffrey2)) {
            System.out.println("Same String ||" + geoffrey + "||" + geoffrey2 + "|");
        }}}
```

Reading the keyboard

- `System.in` is, by default, set to receive keyboard input
- Use this pattern to read from keyboard
- **the code on this slide will not compile/run**

```
public class Student {
    String name;
    int age;

    public Student(String n, int a) {
        name = n;
        age = a;
    }

    public String toString() {
        StringBuilder sb =
           new StringBuilder("Details.............");
        sb.append("\nName: “).append(this.name);
        sb.append(“\nAge:  ").append(age);
        return sb.toString();
    }
```

```
    public static void main(String[] args) {
        BufferedReader br = new BufferedReader(
                            new InputStreamReader(System.in));
        String name;
        int age;
        System.out.print("Enter student name: ");
        name = br.readLine().trim();
        System.out.print("Enter Age: ");
        age = Integer.parseInt(br.readLine());
        Student student = new Student(name, age);
        System.out.println("\n" + student.toString());
    }
}
```

Exceptions

- Unexpected events during execution
  - unavailable resource
  - unexpected input
  - logical error
- In Java, exceptions are objects
- 2 options with an Exception
  - "Throw" it
    - this says that the exception must be handled elsewhere
  - "Catch" it.
    - handle the problem here and now

Catching Exceptions

- Exception handling
- `try-catch`
- An exception is caught by having control transfer to the matching `catch` block
- If no exception occurs, all `catch` blocks are ignored

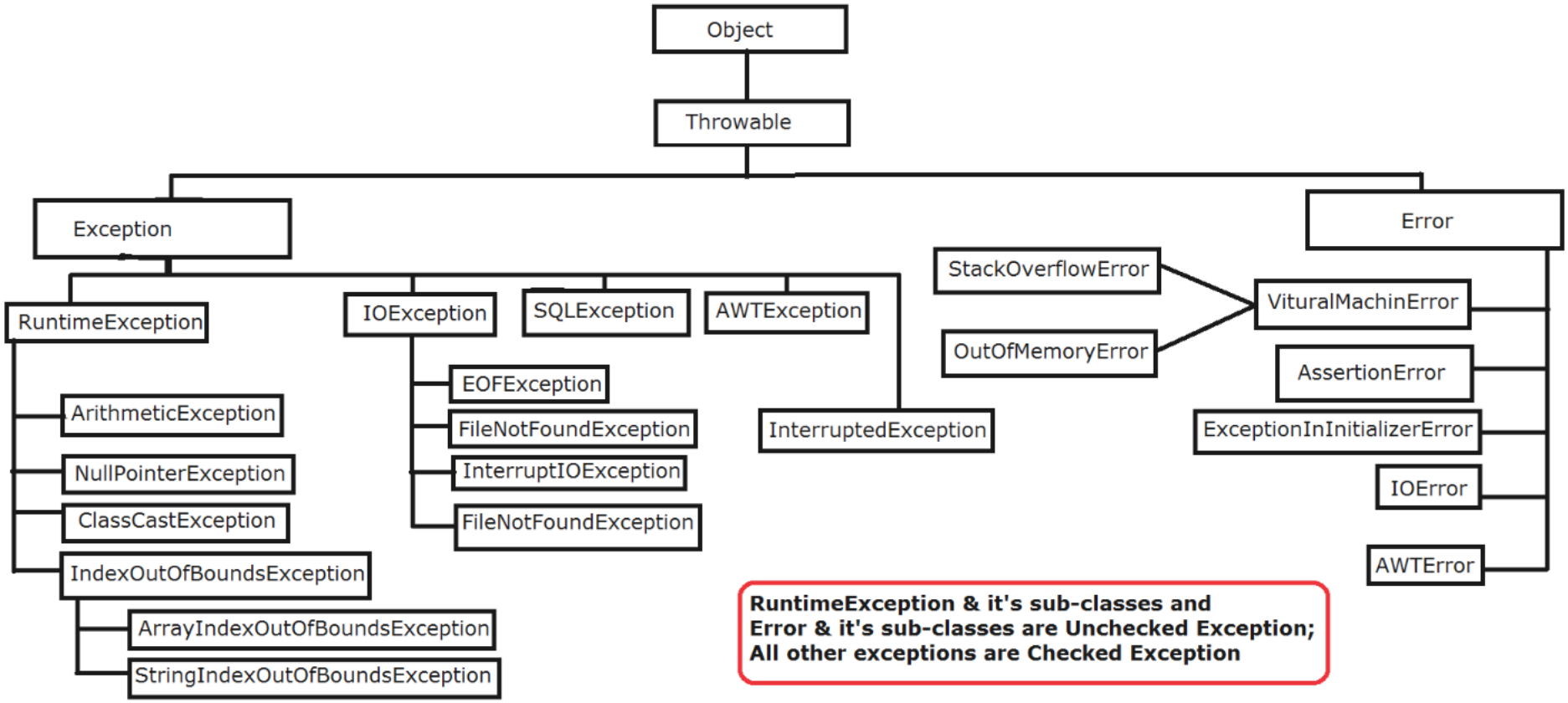
```
try {
    guardedBody
} catch (exceptionType1 variable1) {
    remedyBody1
} catch (exceptionType2 variable2) {
    remedyBody2
} ...
    ...
```

Throwing Exceptions

- An exception is thrown
  - implicitly by the JVM because of errors
  - explicitly by code
- Exceptions are objects
  - throw an existing/predefined one
  - make a new one
- Method signature – `throws`

```
public static int parseInt(String s)
throws NumberFormatException
```

Java's Exception Hierarchy



Handling Exceptions try-catch

```
public static void main(String[] args) {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String name;
        int age;
        try {
            System.out.print("Enter student name: ");
            name = br.readLine().trim();
        } catch (IOException e) {
            System.err.println("problem " + e);
            return;
        }
        try {
            System.out.print("Enter Age: ");
            age = Integer.parseInt(br.readLine());
        } catch (IOException e) {
            System.err.println("problem " + e);
            return;
        } catch (NumberFormatException e) {
            System.err.println("problem " + e);
            return;
        }
        Student student = new Student(name, age);
        System.out.println("\n" + student.toString());
    }
```

Exceptions should be handled as soon as possible.

try-catch should enclose as little code as possible

Handling Exceptions throws

Sometimes it is better to handle exceptions elsewhere ..

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class NameAndAge {
    private String name;
    private int age;
    public void getNameAndAge(InputStream inStream) throws IOException, NumberFormatException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter student name: ");
        name = br.readLine().trim();
        System.out.print("Enter Age: ");
        age = Integer.parseInt(br.readLine());
    }
    public static void main(String[] args) {
        try {
            NameAndAge nameAndAge = new NameAndAge();
            nameAndAge.getNameAndAge(System.in);
            System.out.println("\n" + nameAndAge);
        } catch (IOException ioe) {
            System.err.println("problem " + ioe);
        } catch (NumberFormatException nfe) {
            System.err.println("problem " + nfe);
        }}}
```

Every throw must be caught

Never throw from main

try/catch — with resources

```
public void readOneLineTC(String filename)
 {
     BufferedReader br;
     try {
         br = new BufferedReader(
               new FileReader(filename));
         br.readLine();
     } catch (FileNotFoundException fnf) {
         System.err.println("No file " + e);
     }  catch (IOException e) {
         System.err.println("Reading " + e);
     } finally {
         if (br!=null) {
             try {
                 br.close();
             } catch (IOException ioe) {
                 System.err.println("Close" + ioe);
             }
         }
     }
 }
```

```
public void readOneLineTCR(String filename)
    {
        try (BufferedReader br = new BufferedReader(
                        new FileReader(filename));) {
            br.readLine();
            // close unnecessary in this formulation
        } catch (FileNotFoundException e) {
            System.err.println("Open " + e);
        } catch (IOException e) {
            System.err.println("Reading " + e);
        }
    }
```

finally == code that WILL be executed

Close can throw an exception so it too must be caught

if time, write program to demo try/catch/fianlly

Software Design Goals

- Robustness
  - software capable of error handling and recovery
  - programs should never crash
    - ending abruptly is not crashing
- Adaptability
  - software able to evolve over time and changing conditions (without huge rewrites)
- Reusability
  - same code is usable as component of different systems in various applications
  - The story of Mel — https://www.cs.utah.edu/~elb/folklore/mel.html

OOP Design Principles

- Modularity
  - programs should be composed of "modules" each of which do their own thing
    - each module is separately testable
  - Large programs are built by assembling modules
  - Objects (Classes) are modules
- Abstraction
  - Get to the core — non-removable essence of a thing
  - Most pencils are yellow, but yellowness does not required
- Encapsulation
  - Nothing outside a class should know about how the class works.
    - For instance, does the Object class have any instance variables. (Of what type?)
  - Allows programmer to totally change internals without external effect

OOP Design

- Responsibilities/Independence: divide the work into different classes, each with a different responsibility and are as independent as possible
- Behaviors: define the behaviors for each class carefully and precisely, so that the consequences of each action performed by a class will be well understood by other classes that interact with it.

Class Definition

- Primary means for abstraction in OOP
- Class determines
  - the way state information is stored – via instance variables
  - a set of behaviors – via methods
- Classes encapsulate
  - `private` instance variables
  - `public` accessor methods (getters)

`toString`

- Special method in a class that provides a way to customize printing objects
- returns a `String` representation of the instance object that can be used by
- public String toString()

Student (again)

show in VS Code

Constructors

- Constructors are never inherited
- A class may invoke the constructor of the class it extends via a call to `super` with the appropriate parameters
  - e.g. `super()`
  - `super` must be in the first line of constructor
  - If no explicit call to `super`, then an implicit call to the zero-parameter `super` will be made
- A class make invoke other constructors of their own class using `this()`
  - `this` must be first
  - Cannot explicitly use both `super` and `this`