CS206

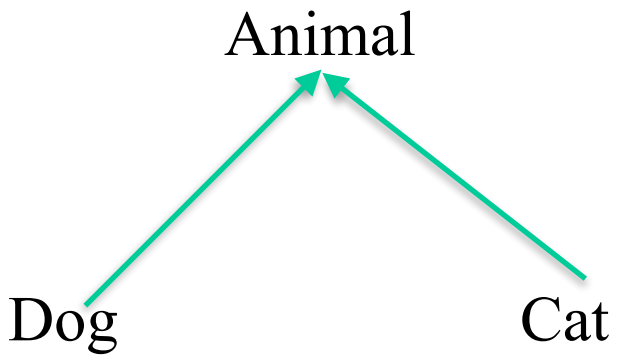
I/O Methods

Files/Exceptions

Inheritance

Casting — continued

- SPCA shelter for only dogs and cats
- Create 3 classes, Dog and Cat that "inherit" from Animal.
- Use array to hold all animals
  - Animal a = new Animal[100];
- But deal with dogs cats separately later



Strings Review

- 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");
```

`String` class methods

- `charAt(int index)`
  - Returns the character at the specified index
- `equals(String anotherString)`
  - Compares a string to a specified object
- `indexOf(char c)`
  - Returns the index value of the first occurrence of a character within the input string
- `indexOf(String str)`
  - Returns the index value of the first occurrence of a substring within the input string
- `length()`
  - Returns the number of characters in the input string
- `substring(int startIndex, int endIndex)`
  - Returns a new string that is part of the input string
- `toLowerCase()`
  - Converts all the characters to lower case
- `toUpperCase()`
  - Converts all the characters to upper case
- `String concat(String anotherString)`
  - Concatenates with anotherString and returns it

Strings, example

```
/************************
 * Author:  G. Towell
 * Created: August 28, 2019
 * Modified: August 29, 2019
 * Purpose:
 *    String Methods sample
************************/
public class Stringer
{
    public static void main(String[] args)
    {
     String geoffrey = "Geoffrey";
     System.out.println(geoffrey);
     String geoff = geoffrey.substring(0, 5);
     System.out.print(geoff + "\n");
     String c = geoffrey.concat(geoff);
     System.out.println("|"+geoffrey+"|"+geoff+"|"+c+"|");
     String d = geoffrey + geoff;
     System.out.println("|"+geoffrey+"|"+geoff+"|"+d+"|");
    }
}
```

Simple Input

- `System.in` and `Scanner` object

```
import java.util.Scanner;                    // loads Scanner definition for our use

public class InputExample {
  public static void main(String[ ] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your age in years: ");
    double age = input.nextDouble( );
    System.out.print("Enter your maximum heart rate: ");
    double rate = input.nextDouble( );
    double fb = (rate − age) ∗ 0.65;
    System.out.println("Your ideal fat-burning heart rate is " + fb);
  }
}
```

`java.util.Scanner` Methods

- Can read from many sources
  - strings, files, keyboard, …
- reads the input and divides it into tokens – strings separated by delimiters

hasNext( ):	Return **true** if there is another token in the input stream.
next( ):	Return the next token string in the input stream; generate an error if there are no more tokens left.
hasNext*Type*( ):	Return **true** if there is another token in the input stream and it can be interpreted as the corresponding base type, *Type*, where *Type* can be Boolean, Byte, Double, Float, Int, Long, or Short.
next*Type*( ):	Return the next token in the input stream, returned as the base type corresponding to *Type*; generate an error if there are no more tokens left or if the next token cannot be interpreted as a base type corresponding to *Type*.

File I/O

1. Create a new `Scanner` object linked to the file we want to read

   ```
   Scanner input = new Scanner(new
   File(<filename>));
   ```

2. Use `hasNextLine()` and `nextLine()` methods to read line by line
3. close()
   1. Every opened scanner on a file must be closed!

File I/O

```
/************************
 * Author:  G. Towell
 * Created: August 28, 2019
 * Modified: August 29, 2019
 * Purpose:
 *    Scanner practice
 *    DOES NOT COMPILE
 ************************/
public class FileScanner
{
    public static void main(String[] args)
    {
         Scanner input;
         String line;
         input = new Scanner(new File("Hello.txt"));

         // read a file line by line, then print line word by word

         input.close();
     }
}
```

Exceptions

- Unexpected events during execution
  - unavailable resource
  - unexpected input
  - logical error
- In Java, exceptions are objects that can be thrown by code expecting to encounter it
- An exception may also be caught by code that will handle the problem

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

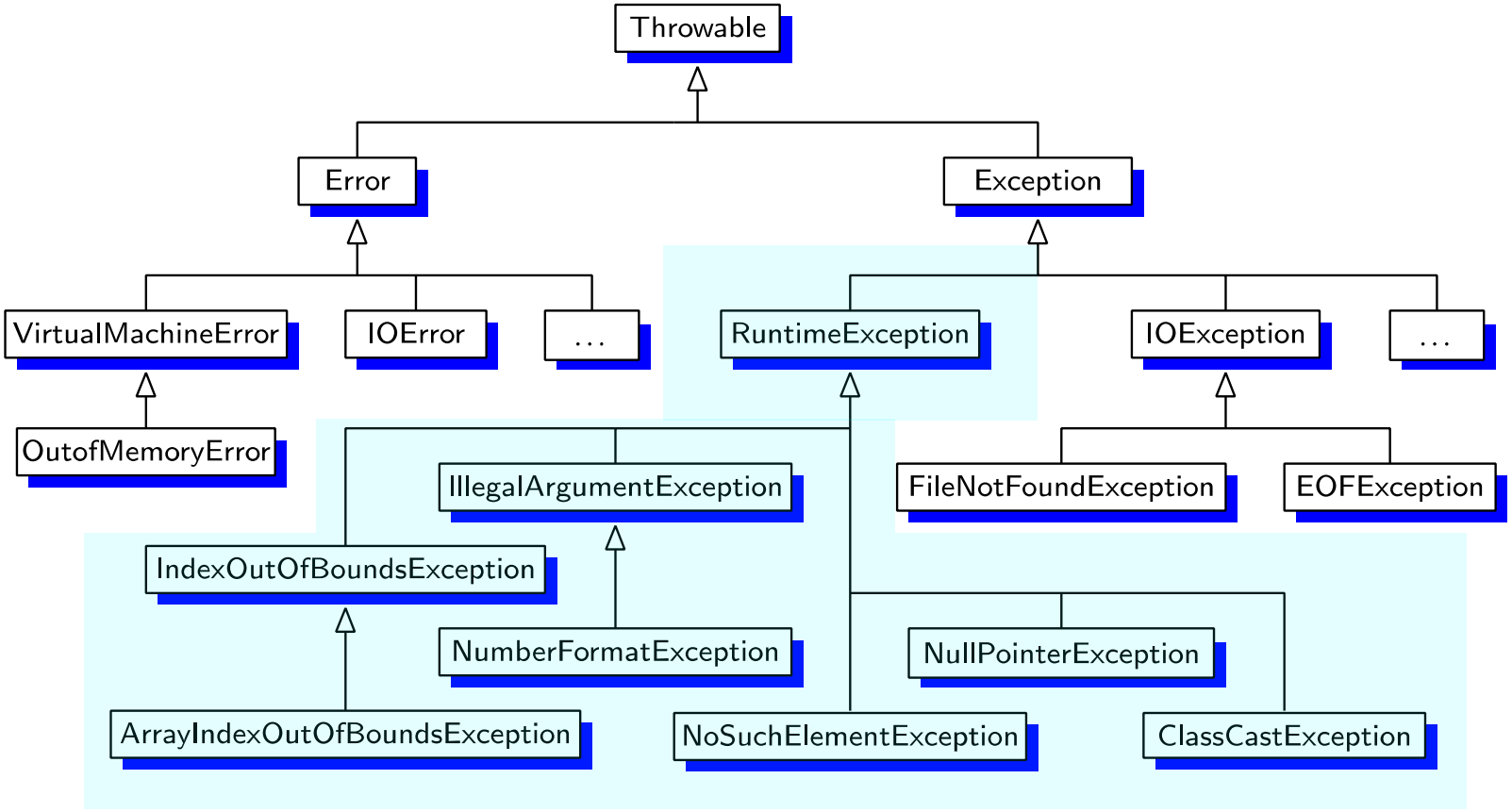
```
try {
    guardedBody
} catch (exceptionType1 variable1) {
    remedyBody1
} catch (exceptionType2 variable2) {
    remedyBody2
} ...
    ...
```

Throwing Exceptions

- An exception is thrown
  - implicitly by the JVM because of errors
  - explicitly thrown 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 Hierachy



Using Exceptions

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
/**********************
 * Author:  G. Towell
 * Created: August 28, 2019
 * Modified: August 29, 2019
 * Purpose:
 *    Scanner & Exceptions practice
 **********************/
public class FileScanner {
    public static void main(String[] args)
    {
         Scanner input;
         String line;
         try {
              input = new Scanner(new File("Hello.txt"));
        }
         catch (FileNotFoundException e) {
              System.out.println("Error in opening the file:" + inFileName);
         }
         finally {
              if (input!=null) input.close();
         }
     }
}
```

Exceptions — with resources

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

/************************
 * Author:  G. Towell
 * Created: August 28, 2019
 * Modified: August 29, 2019
 * Purpose:
 *    Scanner & shortened exceptions example
 ************************/
public static void main(String[] args) {
     String inFileName = "liam.txt";
     String line;
     try (Scanner input=new Scanner(new File(inFileName))) {
     }
     catch (FileNotFoundException e) {
          System.out.println("Error in opening the file:" + inFileName);
          System.exit(1);
     }
}
```

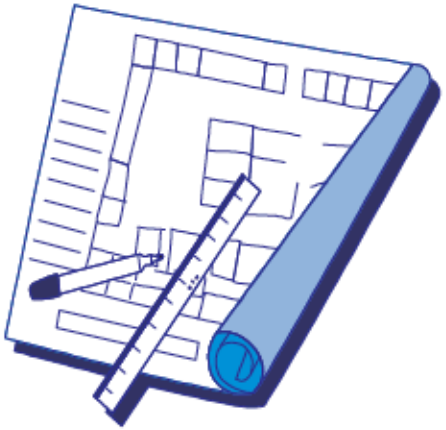
Software Design Goals

- Robustness
  - software capable of error handling and recovery
- 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

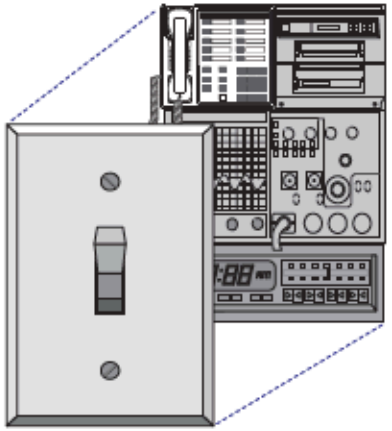
OOP Design Principles

- Modularity
- Abstraction
- Encapsulation

Modularity



Abstraction



Encapsulation

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
- Class encapsulates
  - `private` instance variables
  - `public` accessor methods (getters)

Example

```
class Student {
  private String name;
  private int id;

  public Student(String name, int id) {
    this.name = name;
    this.id = id;
  }

  public String getName() {return name;}
  public int getId() {return id;}
}
```

toString

- Special method in a class that provides a way to customize printing objects

```
Student s = new Student("Ada Lee", 1234);
System.out.println(s); //??
```

- returns a `String` representation of the instance object that is used by `System.out.println`
- `public String toString()`

Student

```
class Student {
  private String name;
  private int id;
  // constructor and getters not shown

  public String toString() {
    return name+" "+id;
  }
}
```

Inheritance

- Allow a new class to be defined based on an existing class
  - Existing: base, super or parent class
  - New: subclass or child class
- Keyword `extends`

```
class CSStudent extends Student{
```

- `CSStudent` inherits all `public` instance variables and methods of `student`

Constructors

- Constructors are never inherited
- A subclass may invoke the superclass constructor via a call to `super` with the appropriate parameters
- If calling `super`, it must be in the first line of the subclass' constructor
- If no explicit call to `super`, then an implicit call to the zero-parameter `super()` will be made

CSStudent

```
class CSStudent extends Student{
  private boolean isMajor;
  public CSStudent(String name, int id, boolean isMajor){
    super(name, id);
    this.isMajor = isMajor;
  }
  public boolean getIsMajor() {return isMajor;}
}
CSStudent s1 = new CSStudent("Pam Chi", 1111, true);
CSStudent s2 = new CSStudent("Di Xu", 2222, false);
System.out.println(s1);
System.out.println(s2);
```

Output

```
Pam Li 1111 is a CS major
Di Xu 2222 is not a CS major
```