CS 106
INTRODUCTION TO DATA STRUCTURES

SPRING 2020

PROF. SARA MATHIESON

HAVERFORD COLLEGE

ADMIN

- **By lab on Friday you should at least be through Section 1 of Lab 1**

Tip: include a toString *method for your class for quick testing*

- **Lab 2 (continuation of Lab 1) will be posted soon so you can get started early**
- **Read the book**
- **Email me if you will miss class, email Suzanne if you will miss lab**
- **If your machine is not working, I will give you a sticky note. Write "not working" and stick it on the machine. Thanks!**

OFFICE HOUR UPDATES

Monday 9-11pm TA hours (Steve)

Tuesday 4:30-6pm Instructor office hours (Sara)

Tuesday 7-9pm TA hours (Emile)

Saturday 4-6pm TA hours (Will)

Sunday 7-9pm TA hours (Juvia)

All in H110!

FEB 4 OUTLINE

- Recap Inheritance and Interfaces
- Abstract classes
- Exceptions
- Arrays
- Loops with arrays + foreach

FEB 4 OUTLINE

- **Recap Inheritance** and Interfaces

  Review quiz (not collected)

- Abstract classes

- Exceptions

- Arrays

- Loops with arrays + foreach

HANDOUT 5 (PAGE 1)

HANDOUT 5 (PAGE 1)

1. This code demonstrates *inheritance*, with `Computer` as a *parent class* and `Mac` as a *child class*. The line with `(Computer) c1` demonstrates *upcasting*, and the line with `(Mac) c3` demonstrates *downcasting*.

HANDOUT 5 (PAGE 1)

1. This code demonstrates *inheritance*, with Computer as a *parent class* and Mac as a *child class*. The line with (Computer) c1 demonstrates *upcasting*, and the line with (Mac) c3 demonstrates *downcasting*.

2. In toString in Mac, we should have said super.toString() to get numFiles. After this is fixed, we get the printout:

```
Num files: 4, Version: Sierra
Sierra
Num files: 4, Version: Sierra
```

HANDOUT 5 (PAGE 1)

1. This code demonstrates *inheritance*, with Computer as a *parent class* and Mac as a *child class*. The line with (Computer) c1 demonstrates *upcasting*, and the line with (Mac) c3 demonstrates *downcasting*.

2. In toString in Mac, we should have said super.toString() to get numFiles. After this is fixed, we get the printout:

```
Num files: 4, Version: Sierra
Sierra
Num files: 4, Version: Sierra
```

3. The last line of main throws an error. Since c3 was originally a Computer, it does not have a getVersion method.

HANDOUT 5 (PAGE 1)

```
public class Computer {

    private int numFiles;

    public Computer(int numFiles) {
        this.numFiles = numFiles;
    }

    public int getNumFiles() {
        return numFiles;
    }

    @Override
    public String toString() {
        return "Num files: " + numFiles;
    }
}
```

HANDOUT 5 (PAGE 1)

```
public class Mac extends Computer {

    private String version;

    public Mac(int numFiles, String version) {
        super(numFiles);
        this.version = version;
    }

    public String toString() {
        // corrected: call super to get the parent string
        // alternative: use getNumFiles() from parent class
        return super.toString() + ", Version: " + version;
    }

    public String getVersion() {
        return version;
    }
}
```

HANDOUT 5 (PAGE 1)

```
public static void main(String[] args) {

    Mac c1 = new Mac(4, "Sierra");
    //Mac c1 = new Mac(4); // note: this does not work
    System.out.println(c1);
    System.out.println(c1.getVersion());

    Computer c2 = (Computer) c1;
    System.out.println(c2);

    Computer c3 = new Computer(10);
    // correction: both of these lines cause runtime errors!
    //Mac c4 = (Mac) c3;
    //System.out.println(c4.getVersion());
}
```

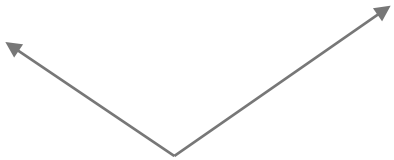
FEB 4 OUTLINE

- **Recap** Inheritance and **Interfaces**
- Abstract classes
- Exceptions
- Arrays
- Loops with arrays + foreach

INTERFACE EXAMPLE

```
1  /** Interface for objects that can be sold. */
2  public interface Sellable {
3
4    /** Returns a description of the object. */
5    public String description();
6
7    /** Returns the list price in cents. */
8    public int listPrice();
9
10   /** Returns the lowest price in cents we will accept. */
11   public int lowestPrice();
12 }
```

```
1  /** Interface for objects that can be transported. */
2  public interface Transportable {
3    /** Returns the weight in grams. */
4    public int weight();
5    /** Returns whether the object is hazardous. */
6    public boolean isHazardous();
7  }
```



Interfaces define a contract of methods that must be implemented by any class implementing the interface

Textbook: Section 2.3

INTERFACE EXAMPLE

```
1  /** Interface for objects that can be sold. */
2  public interface Sellable {
3
4    /** Returns a description of the object. */
5    public String description( );
6
7    /** Returns the list price in cents. */
8    public int listPrice( );
9
10   /** Returns the lowest price in cents we will accept. */
11   public int lowestPrice( );
12 }
```

```
1  /** Interface for objects that can be transported. */
2  public interface Transportable {
3    /** Returns the weight in grams. */
4    public int weight( );
5    /** Returns whether the object is hazardous. */
6    public boolean isHazardous( );
7  }
```

Interfaces define a contract of methods that must be implemented by any class implementing the interface

```
1  /** Class for photographs that can be sold. */
2  public class Photograph implements Sellable {
3    private String descript;                    // description of this photo
4    private int price;                          // the price we are setting
5    private boolean color;                      // true if photo is in color
6
7    public Photograph(String desc, int p, boolean c) {    // constructor
8      descript = desc;
9      price = p;
10     color = c;
11   }
12
13   public String description( ) { return descript; }
14   public int listPrice( ) { return price; }
15   public int lowestPrice( ) { return price/2; }
16   public boolean isColor( ) { return color; }
17 }
```

Classes that implement the interface can also have other methods

Textbook: Section 2.3

INTERFACE EXAMPLE

```
1  /** Class for objects that can be sold, packed, and shipped. */
2  public class BoxedItem implements Sellable, Transportable {
3    private String descript;          // description of this item
4    private int price;                // list price in cents
5    private int weight;               // weight in grams
6    private boolean haz;              // true if object is hazardous
7    private int height=0;             // box height in centimeters
8    private int width=0;              // box width in centimeters
9    private int depth=0;              // box depth in centimeters
10   /** Constructor */
11   public BoxedItem(String desc, int p, int w, boolean h) {
12     descript = desc;
13     price = p;
14     weight = w;
15     haz = h;
16   }
17   public String description( ) { return descript; }
18   public int listPrice( ) { return price; }
19   public int lowestPrice( ) { return price/2;       }
20   public int weight( ) { return weight; }
21   public boolean isHazardous( ) { return haz; }
22   public int insuredValue( ) { return price*2; }
23   public void setBox(int h, int w, int d) {
24     height = h;
25     width = w;
26     depth = d;
27   }
28 }
```

*Multiple inheritance is not allowed, but we *can* implement multiple interfaces*

Textbook: Section 2.3

EXAMPLE FROM MY RESEARCH

```
// interface for all Esteps
public interface Estep {

  public double getEstepLogLikelihood();
  public double[] getExpectedSegments();
}
```

One interface defined, 4 classes that implement it, each with a different algorithm "underneath"

```
public class EstepLinearLol implements Estep {
```

```
public class EstepLinearPac implements Estep {
```

```
public class EstepQuadLol implements Estep {
```

```
public class EstepQuadPac implements Estep {
```

EXAMPLE FROM MY RESEARCH

*Another interface has a method that *must* take in an Estep, but it doesn't care which one.*

```
public interface MstepLinear extends UnivariateRealFunction {

  public void updateEachIter(Estep eStep);
  public void updateParamIdx(int paramIdx, double prevSize);
  public void setDebug(boolean debug);
}
```

EXAMPLE FROM MY RESEARCH

*Another interface has a method that *must* take in an Estep, but it doesn't care which one.*

```
public interface MstepLinear extends UnivariateRealFunction {

  public void updateEachIter(Estep eStep);
  public void updateParamIdx(int paramIdx, double prevSize);
  public void setDebug(boolean debug);
}
```

In main, we can use the methods inside Estep because we know every instance of an Estep will have them.

```
if (printExpectedSegs) {
  System.out.println("expected segments: " + Arrays.toString(eStep.getExpectedSegments()));
}
System.out.println("E-step log likelihood: " + eStep.getEstepLogLikelihood());
```

FEB 4 OUTLINE

- Recap Inheritance and Interfaces
- **Abstract classes**
- Exceptions
- Arrays
- Loops with arrays + foreach

ABSTRACT CLASSES

- Somewhere between a parent class and an interface
- You *cannot* instantiate an instance of an abstract class
- But you *can* implement methods in an abstract class that can be used by child classes
- Use keyword extends, not implements

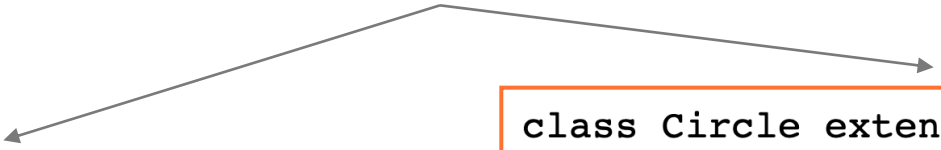
ABSTRACT CLASSES

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

Oracle Java Documentation

ABSTRACT CLASSES

```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```



```
class Rectangle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

```
class Circle extends GraphicObject {
    void draw() {
        ...
    }
    void resize() {
        ...
    }
}
```

Oracle Java Documentation

FEB 4 OUTLINE

- Recap Inheritance and Interfaces
- Abstract classes
- **Exceptions**
- Arrays
- Loops with arrays + foreach

2 MAIN WAYS OF DEALING WITH EXCEPTIONS

1. **Throw an error**

```
public static boolean isPrime(int n) throws IllegalArgumentException {
    if (n > 0) {
        return true; // stub return value
    } else {
        throw new IllegalArgumentException("input" + n + "not positive!");
    }
}
```

2. **try/catch**

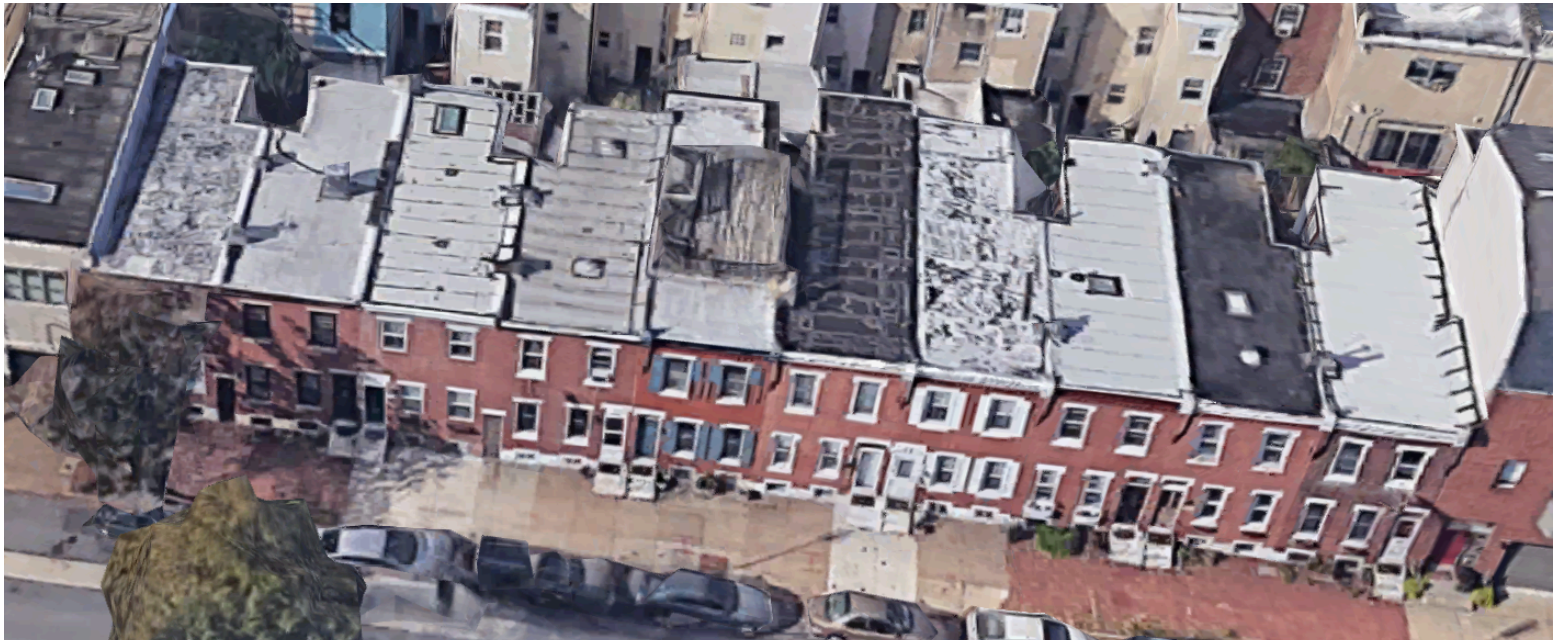
```
try {
    reader = new CSVReaderHeaderAware(new FileReader("compas-scores.csv"));

} catch (FileNotFoundException e) {
    e.printStackTrace();

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

More info in the book: Section 2.4

FEB 4 OUTLINE

- Recap Inheritance and Interfaces
- Abstract classes
- Exceptions
- **Arrays**
- Loops with arrays + foreach

ARRAY INTUITION: HOUSES ALONG A STREET



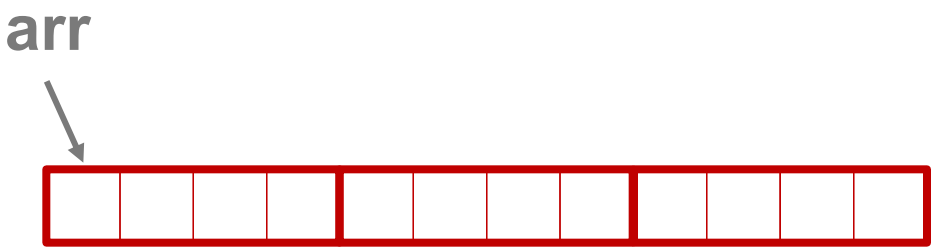
Taney Street in Philadelphia

Images: Google maps

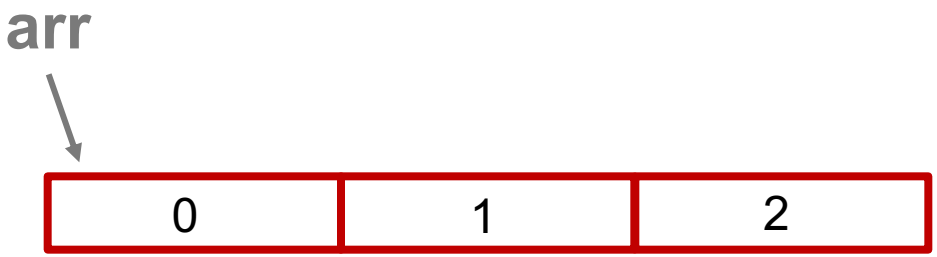
- Houses are 14 feet wide
- Numbers represent distance from the beginning of the street
- These numbers are like addresses in memory!
- Data size (14 ft) is abstracted away
  - Example: postal worker goes house by house, but doesn't need to know the size

Idea of Arrays

Block of memory holding ints (4 bytes each)



- Abstract away the size of the data type
- Then indexing becomes counting!



ID: 555113275

Code | Vocab | Memory

```
int[] digits;
digits = new int[9];
int[] digits = new int[9];
digits[0] = 5;
digits[1] = 5;
digits[8] = 2;
```

declare

allocation

assignment

(re) assignment

"set"

(initialize)

type

allocation

digits

HEAP

ex

memory locations.

52 56 ... 80 84

5 5 0 0 0 0 0 0 2

7

0 1 2 3 4 5 6 7 8

X

5

```
int x = digits[1];
digits[1] = 7;
```

indexing

digits[7] ?

52 + 4 · 7 = 80

52 ↑ start

4 ↑ ints are 4 bytes

7 ↑ index

Arrays

- size known in advance
- continuous block of memory
- cannot extend or downsize
- fast & efficient
- constant time (1 operation) to index

4:30 - 6pm

H 110

① `int[] arr = new int[3];`

`arr[1] = 7;`

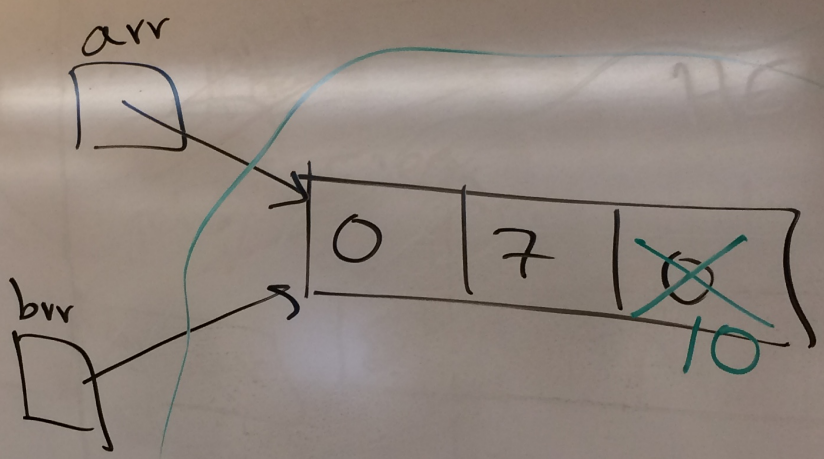
`print(arr);` [0, 7, 0]

`int brr = arr;`

`brr[2] = 10;`

`print(arr);` [0, 7, 10]

copy(arr)



(2)

```
String[] colors = {"red","blue","green"};
print(colors.length);
colors.append("yellow");
print(colors.length);
```

static

Arrays.toString(<myArr>)

myData.toString()

non-static