Compsci 201, L3: Object-Oriented Programming (OOP)
Logistics, Coming up

• This Wednesday, 1/25
  • Interfaces, Implementations, ArrayList data structure
  • First APT set (short programming exercises) due
  • Need to do at least 4 for full credit

• This Friday, 1/27
  • Discussion 2: APTs, Sets, Strings, Git

• Next Monday 1/30
  • Prjoect 0: Person201 due (warmup project)
  • Maps and Sets in class
## Course Website Reminder

Schedule page has slides, recordings, and more

<table>
<thead>
<tr>
<th>Week</th>
<th>Day</th>
<th>Reference</th>
<th>In Class</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M 1/9</td>
<td></td>
<td>No meeting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W 1/11</td>
<td>Z8</td>
<td>L1: What is Computer Science [slides] [code]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F 1/13</td>
<td></td>
<td>No meeting - Setup tech and review</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>M 1/16</td>
<td></td>
<td>No meeting - M.L.K. Jr. Day</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W 1/18</td>
<td>Z1-Z7 (intro Java review) as needed</td>
<td>L2: Intro to Java [slides] [recording]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F 1/20</td>
<td></td>
<td>D1: Algorithmic Problem Solving [discussion document]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>M 1/23</td>
<td>Z9</td>
<td>L3: Object-Oriented Programming</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W 1/25</td>
<td>Z10</td>
<td>L4: Interfaces, Implementations, ArrayList</td>
<td>APT 1</td>
</tr>
<tr>
<td></td>
<td>F 1/27</td>
<td></td>
<td>D2: Java, Git</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M 1/30</td>
<td>Z11</td>
<td>L5: Maps and Sets</td>
<td>P0: Person201</td>
</tr>
</tbody>
</table>
Course Policy Reminders

• Collaboration reminder: Can discuss projects and APTs conceptually, **code must be your own**.
  • If you can’t write the code yourself, you’re not going to be ready for whatever you want to do next.

• Getting Help reminder: We want to help!
  • [Course website getting help page](#)
  • Su-Th *every evening*, Use OhHai to queue
  • Some daytime hours, plus Ed discussion
  • Expect help about your process and how to make progress – not “solutions” or for TAs to debug your code for you.
Person in CS: Grace Hopper

• PhD in math from Yale in 1930s
• Joined Navy Reserve during WW2
• 1940s, began working on developing early computers:
  • Mark 1
  • UNIVAC 1
• 1950s, began work on the earliest “high level” programming languages
  • FLOW-MATIC
  • COBOL – Still in use!
• Annual Grace Hopper Celebration of Women in Computing, usually in the Fall. Consider attending!

USS Hopper
Intro Java Wrap-up
WOTO

Go to duke.is/v3y2y

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
L02-WOTO2-MoreJava

* Required

* This form will record your name, please fill your name.

Enter your answer

1

NetID *

2

What kind of method is **not** called on an object? *

- static methods
- dynamic methods
- public methods
- private methods
What will be printed by the following program?

```java
public static void main(String[] args) {
    String message = "to be or not to be";
    String[] words = message.split(regex: " ");
    ArrayList<String> list = new ArrayList<>();
    int count = 0;
    for (String w : words) {
        if (list.contains(w)) {
            count++;
        }
        else {
            list.add(w);
        }
    }
    System.out.println(count);
}
```
A static method of the ArrayList class

A dynamic method of the ArrayList class

A static method of the String class

A dynamic method of the String class

The contains method on line 10 is...

```java
ArrayList<String> list = new ArrayList<>();
int count = 0;
for (String w : words) {
    if (list.contains(w)) {
        count++;
    }
    else {
        list.add(w);
    }
```
We say that the contains method on line 10...

```java
ArrayList<String> list = new ArrayList<>();
int count = 0;
for (String w : words) {
    if (list.contains(w)) {
        count++;
    }
} else {
    list.add(w);
}
```

- Is called on w and takes list as an argument
- Is called on list and takes w as an argument
- Is called on both list and w
Comments on Java style

Code blocks:

• Opening { ends first line of if, for, while, or method
• Indent every line inside the block
• Closing } on a separate line, last of block, not indented

```java
int i = 0;
while (i < numbers.length) {
    System.out.println(numbers[i]);
    i++;
}
```

Variable & method names:

• One-word names: lowercase
  ```java
  int index = 0;
  ```
  
• Multi-word names: camelCase
  ```java
  int maxSize = 10;
  ```
  
• Should be informative
More comments on Java style

Class names:
• Capitalized & CamelCase
• MUST match name of .java file!

Comments:
• // for one line
• /* ... */ for multiple lines

```java
public class MethodExample {
    // one line comment
    /* a
    block
    comment */
```
Javadoc: Advanced comments

Person201 query = new Person("Fain", "LSRC", 1);

Person201(Person201(String name, String building, int floor)

Construct Person201 object with information

• Parameters:
  • name preferred name/nickname of person or anonymous
  • building common name of building where you can be found
  • floor which floor is your room
Writing Javadoc

/**
 * Construct Person201 object with information
 * @param name preferred name/nickname of person or anonymous
 * @param building common name of building where you can be found
 * @param floor which floor is your room
 */

public Person201(String name, String building, int floor) {
    myName = name;
    myBuilding = building;
    myFloor = floor;
}

Common annotations for methods include: @param, @returns, @throws
Java printing

```java
String month = "January";
int day = 18;
System.out.println(month);
System.out.println(day);
System.out.printf("Today is %s %d\n", month, day);

int[] array = {1, 2, 3};
System.out.println(array);
System.out.println(Arrays.toString(array));
```

- `String month = "January";` declares a string variable `month` with the value "January".
- `int day = 18;` declares an integer variable `day` with the value 18.
- `System.out.println(month);` prints the value of `month` to the console.
- `System.out.println(day);` prints the value of `day` to the console.
- `System.out.printf("Today is %s %d\n", month, day);` prints the formatted string "Today is {0} {1}\n" with the values of `month` and `day`.
- `int[] array = {1, 2, 3};` declares an integer array `array` with the elements 1, 2, and 3.
- `System.out.println(array);` prints the array `array` to the console.
- `System.out.println(Arrays.toString(array));` prints the string representation of the array `array` to the console.

Values for placeholders:
- `%s` represents a string
- `%d` represents an integer
Debugging

• What is the first line of my program where something is different than I expect?

• Need to see program state during execution.
  • Add print statements to code, see values of variables
    • Will show up in terminal locally, or on APT server

• Use a debugger tool, built into VS Code
  • See the documentation here
Java API HashSet

An import statement:  
```java
import java.util.HashSet;
```

More on HashSet later, but the basics:
• Generic to specify type, does not store duplicates
• Uses add(), size(), contains()

```java
4  public static void main (String[] args) {
5       HashSet<String> strSet = new HashSet<>();
6       strSet.add("Hello");
7       strSet.add("World");
8       strSet.add("Hello");
9
10      if(strSet.contains("World")) {
11          System.out.println(strSet.size());
12      }
```
API Documentation

Reading documentation is an important skill:

docs.oracle.com/en/java/javase/17/docs/api
Object-Oriented Programming
Java is object-oriented

- A language is **object-oriented** if programs in that language are organized by the specification and use of objects.
- “An **object** consists of some internal data items plus operations that can be performed on that data.” – ZyBook

```java
public class StaticUniqueWords {
    public static void main(String[] args) throws IOException {
        Scanner s = new Scanner(new File( pathname: "data/kjv10.txt"));
        HashSet<String> set = new HashSet<>();
        int wcount = 0;
        double start = System.nanoTime();
        while (s.hasNext()) {
            wcount += 1;
            String word = s.next();
            set.add(word);
        }
    }
}
```

We call these **methods**

Scanner is a **Class**, s is an object. Keeps track of where it is in the file and can get the next word.
Aside: Python uses objects too

```python
# pyobject.py
1  s = "Hello World"
2  words = s.split(" ")
3  print(words)
```

Split is a *method* we are calling on this String object, not a regular function!

Same syntax in Python and Java for method calls:
```
<object>.<method>(<method_arguments>)
```
Object Concept

Consider points in two-dimensions.

Class is a blueprint for these objects

Data (instance variables)
- x-coordinate (x)
- y-coordinate (y)

Operations (methods)
- Create a point
- Print a point
- Change coordinates
- Get distance to another point

Each point object has its own x and y value.

All different objects, but each of the same class

Methods should be able to operate on a particular point
Language History: A story of increasing abstraction and organization

**Imperative Programming (Fortran I, etc.)**
- Code organized into a linear sequence of operations.
- All data accessible as variables in the same *global* scope.

**Procedural Programming (C, etc.)**
- Procedures or functions, that can be *called* by a main program.
- Local versus global variables.

**Object-Oriented Programming (Java, etc.)**
- Define more complex variable types using *classes*, use to create *objects*.
- *Dynamic methods* to go along with specific classes/types.
Classes and objects

Class specifies the data and operations for a type of object. They are a template or a blueprint for objects. Alternately, objects are instances of a class.

```java
public class Point {
    public double x;
    public double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

- **Instance variables.** Each Point object has its own x and y value.
- **A constructor** method specifies how to create a new Point object. Same name as class.
- **this** keyword refers to object on which method is called.
- **.** operator accesses instance variable or method of this object.
Creating objects, calling methods

```java
10 public void printPoint() {
11     System.out.printf("(%1f, %1f)%n", x, y);
12 }

14 public static void main(String[] args) {
15     Point p = new Point(-2.0, 2.0);
16     Point q = new Point(1.0, 1.0);
17     p.printPoint();
18     q.printPoint();
19 }
```

Note how the `printPoint()` method “knows” the correct value for x and y – they are stored with the objects on which we call the method as **instance variables**.
Two reasons to call a method

For the **side effect**, what it did to the object

```java
public static void main(String[] args) {
    HashSet<String> strSet = new HashSet<>();
    strSet.add("Hello");
    strSet.add("World");
    strSet.add("Hello");

    if(strSet.contains("World")) {
        System.out.println(strSet.size());
    }
}
```

For the **return value**, no change to object
== or .equals()?

- For primitive types: == checks for equal values.
- For objects, == generally does not.
- Need to use .equals() method for objects.
  - Correct way to compare String objects.
  - Must be implemented for the given Class!

```java
public class Point {
    public double x;
    public double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        Point p = new Point(0.0, 0.0);
        Point q = p;
        Point r = new Point(0.0, 0.0);
        System.out.println(p == q); // true
        System.out.println(p == r); // false
    }
}
```
Default Object .equals

```java
/*
@Override
public boolean equals(Object o) {
    Point other = (Point) o;
    if ((this.x == other.x) && (this.y == other.y)) {
        return true;
    }
    return false;
}
*/
```

Prints false, is just checking memory locations
Overriding default Object .equals

```
@Override
public boolean equals(Object o) {
    Point other = (Point) o;
    if ((this.x == other.x) && (this.y == other.y)) {
        return true;
    }
    return false;
}
```

```
public static void main(String[] args) {
    Point p = new Point(0.0, 0.0);
    Point r = new Point(0.0, 0.0);
    System.out.println(p.equals(r));
}
```

Prints true, is using the method we wrote to check values.
Object vs. object, Inheritance?

• Object: ancestor of all classes
  • Default behavior that's not too useful, ...
  • @Override for .equals

• object – synonym for instance of a class
  • What you get when you call new

• Inheritance is a major topic in object-oriented programming
How do I know what `.equals` does for Java API classes?

Read at the Java API documentation!!!

docs.oracle.com/en/java/javase/17/docs/api

```java
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

equals

public boolean equals(Object o)

Compares the specified object with this list for equality. Returns `true` if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are `equal`. (Two elements `e1` and `e2` are `equal` if `(e1==null ^ e2==null : e1.equals(e2))`.) In other words, two lists are defined to be equal if they contain the same elements in the same order.
```
When do I need new? Every time I create an object, not automatic!

```java
public class Point {
    public double x;
    public double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
}

public static void main(String[] args) {
    Point[] pointArray = new Point[5];
    System.out.print(pointArray[0].x);
}
```

We created the array, but did not call `new` for the individual Point objects.

Exception in thread "main" java.lang.NullPointerException: Cannot read field "x" because "pointArray[0]" is null at Point.main(Point.java:11)
When do I need new again? For every object you want to create!

An even stranger error... creating one object but multiple references to it.

```java
public static void main(String[] args) {
    ArrayList<Point> myPoints = new ArrayList<>();
    Point p = new Point(0.0, 0.0);
    myPoints.add(p);
    p.x = 2.0;
    myPoints.add(p);

    for (Point q : myPoints) {
        q.printPoint();
    }
}
```

Prints

```
(2.0, 0.0)
(2.0, 0.0)
```
Creating a List of points; contains uses equals

```java
import java.util.ArrayList;

public class Point {
    public double x;
    public double y;
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public static void main(String[] args) {
        ArrayList<Point> pointList = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            pointList.add(new Point(0.0, 0.0));
        }
        Point p = new Point(0.0, 0.0);
        System.out.println(pointList.contains(p));
    }
}
```

Good, we called new for every Point object we want to create.

Prints false. ArrayList.contains loops over list checking .equals(), but only default implementation here!
WOTO

Go to duke.is/g6hhd

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
* Required

* This form will record your name, please fill your name.

[NetID]

1.

What is the best place to find reliable information about how a method works for a Java API class? *

- Reddit
- Stackoverflow
- Random googling
- Java API Documentation
Consider the following Point class. On which line is the constructor declared?

- Line 1
- Line 4
- Line 21
- There is no constructor
Same code. When the increaseX method called on line 24 executes, what will **this** refer to?

- The Point class
- The Point object that p refers to
- The Point object that q refers to
- None of the above
Consider the Blob class shown below as well as the BlobDriver class containing a single psvm method. What will be printed by that psvm method on line 16 of BlobDriver? *

```java
public class Blob {
    public String color;
    public String shape;
    public Blob(String color, String shape) {
        this.color = color;
        this.shape = shape;
    }

    @Override
    public boolean equals(Object obj) {
        Blob other = (Blob) obj;
        if (other.shape.equals(this.shape)) {
            return true;
        }
        return false;
    }
}
```

```java
import java.util.ArrayList;
public class BlobDriver {
    public static void main(String[] args) {
        ArrayList<Blob> myBlobs = new ArrayList<String>();
        String[] colors = {"red", "white"};
        String[] shapes = {"round", "obli"};
        for (String color : colors) {
            for (String shape : shapes) {
                Blob newBlob = new Blob(color, shape);
                if (!myBlobs.contains(newBlob)) {
                    myBlobs.add(newBlob);
                }
            }
        }
        System.out.println(myBlobs.size());
    }
}
```

- 0
- 3
- 4
- 7
- 12
Public vs. Private

- **Public** – Can be accessed by code outside of the class.

- **Private** – Can only be accessed by code inside of the class.

### Code Example

**Record.java**

```java
public class Record {
    public String displayName;
    private int uniqueID;

    public Record(String name, int id) {
        displayName = name;
        uniqueID = id;
    }
}
```

**PublicPrivate.java**

```java
public class PublicPrivate {
    public static void main (String[] args) {
        Record rec = new Record("Fain", 12345);
        System.out.println(rec.displayName);
        System.out.println(rec.uniqueID);
    }
}
```
The value of privacy

Suppose your entire system crashes terribly if some code is called on a negative uniqueID.

```java
public class Record {
    public String displayName;
    private int uniqueID;

    public Record(String name) {
        displayName = name;
    }

    public void setID(int id) {
        if (id < 0) {
            System.out.println("Must be nonnegative");
        } else {
            uniqueID = id;
        }
    }
}
```

uniqueID is private, so other code cannot directly change it

Can check for correctness in only code allowed to change uniqueID
(Im)mutability

• An object is immutable if you cannot change it after creation. Methods that change objects are called mutators.

• Java Strings are immutable, even though you can “append” to them. Creates a new String and assigns it every time!

```java
String s = "Hello";
s += " World";
More like
String s0ld = "Hello";
String sNew = "" + s0ld + " World";

(and then get rid of s0ld)
```
Static belongs to the class

• Regular instance variables and methods are called on an object.

• Static methods are called on the class, do not use any instance variables. Often utility “functions”

```java
public class StaticExample {
    public static void main(String[] args) {
        String s = "Hello World!";
        System.out.println(s.split(" ")[0]);
        System.out.println(Math.sqrt(4.0));
    }
}
```

Note that split is called on a String object

Whereas sqrt is called on the Math class
PSVM: Public Static Void Main Method that is:

- **public** – can call outside of class
- **static** – belongs to class, not an object
- **void** – no return value
- **main** – starting point for a program to run

`args` allows for command-line arguments
APT and OOP, making a PSVM method

Suppose you’re working on the **SandwichBar APT**.

```java
public class SandwichBar {
    public int whichOrder(String[] available, String[] orders){
        // fill in code here
        return 0;
    }
}
```

Remember what you know about Java OOP:

- **whichOrder** is a regular method, need to call on an *object* of the SandwichBar class.
- **whichOrder** has parameters, need to supply those.
- All java programs must begin in a PSVM method.
APT and OOP, making a PSVM method

```java
public class SandwichBar {
    public int whichOrder(String[] available, String[] orders) {
        // fill in code here
        return 0;
    }
}

public static void main(String[] args) {
    String[] testAvailable = { "ham", "cheese", "mustard" };
    String[] testOrders = { "ham cheese" };
    SandwichBar testInstance = new SandwichBar();
    int testResult = testInstance.whichOrder(testAvailable, testOrders);
    System.out.println(testResult);
}
```

PSVM method can be in the same class or in a separate “driver” class in the same directory.

Creating test parameters, using example from APT site.

Make a SandwichBar object

Call the method
Why use Classes/objects?

- Because you must in Java
- Formal specification for complex data structures
- Convenience and ease of correct programming
- Composition, Interfaces, & Implementations, Extending & Inheritance – More later!

It’s ok to not be fully “convinced” yet. But OOP has proven itself to be a powerful paradigm for designing complex scalable software.