

[Textbook Links: UW Library](#)
[Core Java, Volume I - Fundamentals](#) 978-0-13-516630-7
[Core Java for the Impatient](#) 978-0-13-469472-6
[Effective Java for the Impatient](#) 978-0-13-468599-1

Chapter 3

-Bruh, big review
 -Mindmap: [DHH- Java Terms - GitMind](#)
 -OOP in general: [OOP Concepts - GitMind](#)

Variables

-int i;
 -i is a name for a memory location.
 -Variables are stored in the call stack. Objects are stored in the heap.
 -If we make a name (let's say) Student class. The declaration going into the call stack and is initialized as a value. The object is then sent to the heap.
 -Other objects stored in the class are sent to the heap as well, and the orig object points to the new heap memory address.

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

Lincoln Learning 2

-Lambdas

```

// Consumer
Consumer<String> printItem = n -> System.out.println(n);
students.forEach(printItem);

// Function
Function<Integer, Integer> doubly = n -> n * n;
System.out.println(doubly.apply(10));

// Predicate
IntPredicate isDivByThree = n -> n % 3 == 0;
System.out.println(isDivByThree.test(10));

// Supplier
Supplier<Double> randomNumUnder100 = () -> Math.random() * 100;
System.out.println(randomNumUnder100.get());
    
```

(IntelliJ likes putting "other:" and "value:", big gay)

-java.util.Optional
 -A class that holds variables
 -isPresent(); checks if the variable stored is null.
 -Avoids NullPointerExceptions

-Stream

```

// Create a stream from other collection types
//
// Array
String[] shoppingArray = new String[]{"apples",
    "bananas", "cherries", "coffee"};
Stream<String> shoppingArrayStream = Arrays.stream(shoppingArray);

// Lists
List<String> shoppingList = List.of("apples",
    "bananas", "cherries", "coffee");
Stream<String> shoppingListStream = shoppingList.stream();

// For loop in one line
shoppingListStream.forEach(System.out::println);
shoppingListStream.parallelStream().forEach(System.out::println);

// Match
boolean isOnList = shoppingListStream()
    .anyMatch(item -> item.contains("apples"));
    
```

```

// Filter
Stream<String> itemsInAisle = shoppingListStream()
    .filter(item -> item.startsWith("c"));

// Map
List<Integer> numbersList = List.of(4, 2, 6, 9, 10, 17, 3);
Stream<Integer> doubledStream = numbersList.stream()
    .map(n -> n * 2);

// Collect
List<Integer> doubledList = numbersList.stream()
    .map(n -> n * 2)
    .collect(Collectors.toList());
    
```

(left is create, right is usage)

-"var"

```

public static void main(String[] args) {
    var orderedFoodItems = List.of("hot dog", "chips");
    var total = 38.23;
}
    
```

-Shorter Switch

```

int stage = 2;
String season = switch(stage) {
    case 0 -> "Spring";
    case 1 -> "Summer";
    case 2 -> "Fall";
    case 3 -> "Winter";
    default -> {
        System.out.println("This season is invalid");
        yield "Invalid stage";
    }
};
    
```

```

enum Season {
    Spring, Summer, Fall, Winter
}

public static void main(String[] args) {
    Season season = Season.Spring;

    String weather = switch(season) {
        case Spring, Summer -> "Sunny";
        case Fall -> "Rainy";
        case Winter -> "Snowy";
        default -> {
            System.out.println("Invalid season");
            yield "Invalid weather";
        }
    };

    System.out.println(weather);
}
    
```

(right is enum usage)

-String.split()

-Splits a string into a string[]

```

String[] sentences = text.split(regex: "\\.");

System.out.println("The text has " + sentences.length + " sentences.");

String[] words = text.split(regex: "\\s+");
System.out.println("The text has " + words.length + " words.");

System.out.println(Arrays.asList(words));
    
```

-Stack

-Like a deck of cards where only the top can be edited.

-Queue

- Like a line at the supermarket where you can only add to the back, and remove the beginning.
- Queue is declared with LinkedList at data. It's a LinkedList with more rules of how you can manipulate data.

-HashMap

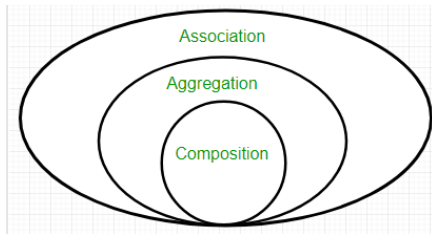
- Like an x&y graph where each key has a corresponding value.
- You want multiple values for a key? Make the value an Array/List.

10/15:

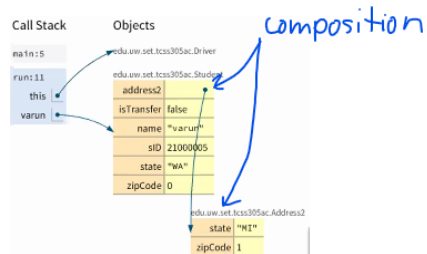
-Composition > Inheritance??? [Composition in Java - GeeksforGeeks](https://www.geeksforgeeks.org/composition-in-java/)

-The composition is achieved by using an instance variable that refers to other objects. If an object contains the other object and the contained object cannot exist without the existence of that object, then it is called composition. In more specific words composition is a way of describing reference between two or more classes using instance variable and an instance should be created before it is used.

-gist: Using another class to make a class (so the class could be a field). One cannot exist without the other like inheritance.



-Composition with memory



-Obvi, you can combine the 2 bruh.

-(REMEMBER) Inheritance constructor starts with "super();"

10/20:

-main (String[] args) {

- Command-line arguments stored as String[]
- Probably needs 2 cases to account for when there are no args.

-Argument vs Parameter

-"Generally a parameter is what appears in the definition of the method. An argument is an instance passed to the method during runtime."

```
public class Example {
    // the variables a and b are parameters
    public static int multiply(int a, int b)
    {
        return a + b;
    }

    public static void main(String[] args)
    {
        int x = 2;
        int y = 5;
        int sum = multiply(x, y);
        System.out.println("SUM IS: " + sum);
    }
}
```

(arguments are runtime)

-Classes

-Interfaces is how you implement multiple interfaces

-You can think of it like a PC having an HDMI interface, where it is not be all end all if methods are not implemented.

-Abstract

- Can't instantiate.
- Helps organization and hierarchy.
- Ex: ChessPieces is the super abstract class of Queen, Bishop, Pawn, etc.
- ChessPiece can't be instantiated, but use polymorphism and get big juice, simplifying parameters and methods.

-With


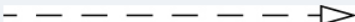




-IntelliJ has a wizard that makes classes. Bruh.

10/22:

-

-Interface VS Inheritance

-UML

Relationship	UML Connector
Inheritance	
Interface implementation	
Dependency	
Aggregation	
Association	
Directed association	

-1 next to diamond is single aggregation. * next to diamond is multiple

-Initialization Block (not common)

-Always executes on object init

```
class Employee
{
    private static int nextId;
    private int id;
    private String name;
    private double salary;
    // object initialization block
    {
        id = nextId;
        nextId++;
    }
    public Employee(String n, double s)
    {
        name = n;
        salary = s;
    }
    public Employee()
    {
        name = "";
        salary = 0;
    }
    . . .
}
```

-ArrayStore Error

-Ex: Manager extends Employee

To make sure no such corruption can occur, all arrays remember the element type with which they were created, and they monitor that only compatible references are stored into them. For example, the array created as `new Manager[10]` remembers that it is an array of managers. Attempting to store an `Employee` reference causes an `ArrayStoreException`.

-final method

-A final method in a superclass can not be overridden by subclasses.

-Casting

You can cast only within an inheritance hierarchy.

Use `instanceof` to check before casting from a superclass to a subclass.

Category	Inheritance	Interface
Description	Inheritance is the mechanism in java by which one class is allowed to inherit the features of another class.	Interface is the blueprint of the class. It specifies what a class must do and not how. Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
Use	It is used to get the features of another class.	It is used to provide total abstraction.
Syntax	class subclass_name extends superclass_name { }	interface <interface_name>{ }
Number of Inheritance	It is used to provide 4 types of inheritance. (multi-level, simple, hybrid and hierarchical inheritance)	It is used to provide 1 types of inheritance (multiple).
Keywords	It uses extends keyword.	It uses implements keyword.
Inheritance	We can inherit lesser classes than Interface if we use Inheritance.	We can inherit enormously more classes than Inheritance, if we use Interface.
Method Definition	Methods can be defined inside the class in case of Inheritance.	Methods cannot be defined inside the class in case of Interface (except by using static and default keywords).
Overloading	It overloads the system if we try to extend a lot of classes.	System is not overloaded, no matter how many classes we implement.
Functionality Provided	It does not provide the functionality of loose coupling	It provides the functionality of loose coupling.
Multiple Inheritance	We cannot do multiple inheritance (causes compile time error).	We can do multiple inheritance using interfaces.

Objects VS Classes

- Object is an instance of a Class.
- Class is the blueprint to make Objects.

10/25:

-Java libs are designed to be reusable.

- You can use Collections and Comparable, and tell them how you want to be compared and how to check the ranking of special Objects.
- equals(), compareTo()

-Object

- The superclass of all objects

-Interfaces

- When overriding, you can't reduce the visibility of methods in the subclass.

10/27:

-Sort

- Very common to implement Comparable and Comparator.
- Comparable
 - compareTo() (aka Natural Ordering)
 - implements Comparable<T> → compareTo(T o)
 - implements Comparable → compareTo(Object o)
 - implements Comparable<Student> → compareTo(Student o) which forgos casting.
 - returns an int, where (this.value - other.value), so that if this is greater, it returns positive.
 - Alternatively, return 1, 0, or -1 if you can't do above.
- Comparator
 - compare(), which you can can ignore compareTo() and define how you want objects compared as well as have multiple.

-Arrays.sort()

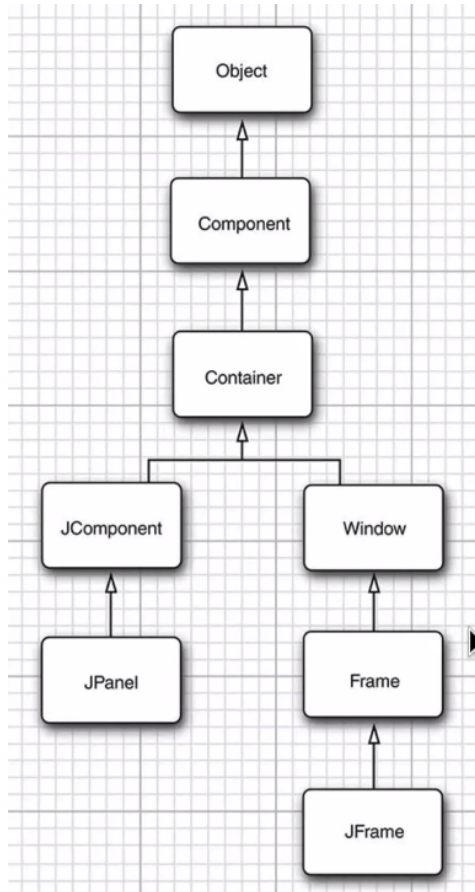
- Doesn't sort primitives. You need to convert it to Wrapper class or something.
- Arrays.sort(array, new customComparator()) will sort it based on the Comparator's results.

-Java is safe

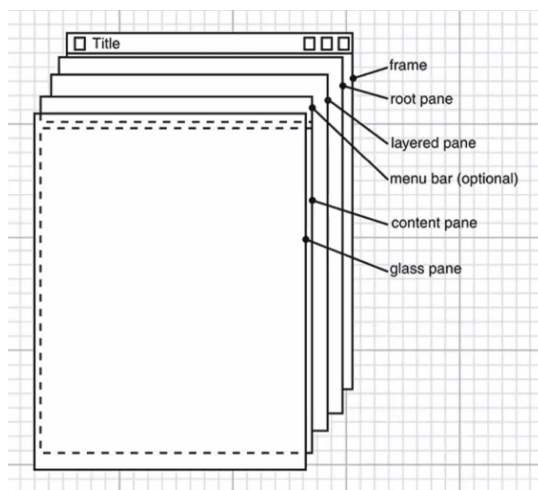
10/29:

-GUI

- SWT (java.awt.*) then came Swing (javax.swing.*)
 - SWT is worse than Swing
 - SWT using OS to format window
 - SWT has less options and less performance
- AWT libs



-JFrame Elements



-Simple frame

Figure 10.1 The simplest visible frame

Listing 10.1 simpleframe/SimpleFrameTest.java

[Click here to view code image](#)

```

1 package simpleFrame;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 /**
7  * @version 1.34 2018-04-10
8  * @author Cay Horstmann
9  */
10 public class SimpleFrameTest
11 {
12     public static void main(String[] args)
13     {
14         EventQueue.invokeLater(() ->
15         {
16             var frame = new SimpleFrame();
17             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18             frame.setVisible(true);
19         });
20     }
21 }
22
23 class SimpleFrame extends JFrame
24 {
25     private static final int DEFAULT_WIDTH = 300;
26     private static final int DEFAULT_HEIGHT = 200;
27
28     public SimpleFrame()
29     {
30         setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
31     }
32 }
  
```

-EventQueue class

-Controls timing of when statements execute.

-Add them components

Listing 10.2 notHelloWorld/NotHelloWorld.java

[Click here to view code image](#)

```
1 package notHelloWorld;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 /**
7  * @version 1.34 2018-04-10
8  * @author Cay Horstmann
9  */
10 public class NotHelloWorld
11 {
12     public static void main(String[] args)
13     {
14         EventQueue.invokeLater(() ->
15         {
16             var frame = new NotHelloWorldFrame();
17             frame.setTitle("NotHelloWorld");
18             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19             frame.setVisible(true);
20         });
21     }
22 }
23
24 /**
25  * A frame that contains a message panel.
26  */
27 class NotHelloWorldFrame extends JFrame
28 {
29     public NotHelloWorldFrame()
30     {
31         add(new NotHelloWorldComponent());
32         pack();
33     }
34 }
35
36 /**
37  * A component that displays a message.
38  */
39 class NotHelloWorldComponent extends JComponent
40 {
```

11/1:

-Git

-Version history/control

-JUnit

-Test small pieces of code

-Like every method of an object

-Pair it with limit testing.

-Ex: add(int x, int y)

x	y	x+y	notes	check
MAX	+	Over MAX	prevent	if (y > MAX - x)
MIN	-	Over MIN	prevent	if (y < MIN - x)
+	+	Over MAX	prevent	if (y > MAX - x)
-	-	Over MIN	prevent	if (y < MIN - x)
+	-	+ or -	trivial	
-	+	+ or -	trivial	
0	0	0	trivial	
0	+	+	trivial	
0	-	-	trivial	
+	0	+	trivial	
-	0	-	trivial	
MAX	MIN	0	trivial	
MIN	MAX	0	trivial	

-Swing

-Text entry (extends from JTextComponent)

-JTextField

-JPasswordField

-JTextArea (multiple lines)

-JScrollPane is useful to add ur JTextArea into.

-getText(), is/setEditable()

-Add them into the panel.

-Buttons

-

-checkboxes: JCheckBox

-radio:

-list of choices/drop down: JComboBox

-sliders: JSlider

11/3:

-It's good to break down each component of the required coding task. It can be used as a checklist and also help assign different tasks to a group.

-Use layouts (ie grid) to format your components in panel(panel.addLayout())

-Button needs action listeners to get when they are clicked.

11/5:

-Layout Manager

-Flow Layout

-Basic layout, its placed next to each other and wraps to next line.

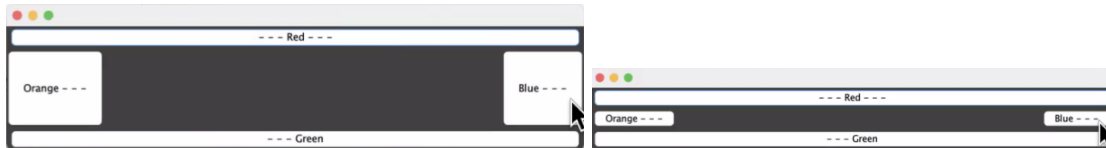
-Grid Layout

-nxm. Evenly scales the size to fit grid

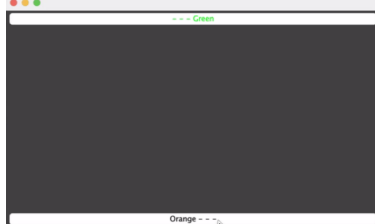
-Border Layout

-Positions the components to the 4 borders of the window.

-It's weird though



-If you add buttons to the same side, it overlaps each other



-A fix is to make another panel, add the buttons to the panel, then the panel goes into the border layout panel.

-Add panels on top of panels on top of panels with different layouts = ez window formatting.

11/8:

-Layout Manager is packaged in awt, but it also works with Swing.

-Event Handling is how your GUI elements can react to keys/mouse. Back in the boomer days, having too many listeners slows down the performance.

-JMenuBar creates a dropdown menu pinned to the top of the window.

11/10:

-Mnemonics are KeyListeners for menus items. JMenuItem.setMnemonic(KeyEvent); (ie KeyEvent.VK_A).

-Common dialogue boxes can are probably in Java already, like saving files. (ie JFileChooser.showSaveDialog(Component parent);)

-There are GUI drafting tools and even Java GUI designing tools.

11/15:

-JFrame and components have layers of panes to display their contents.

-JComponent has a paintComponent(Graphics g) that will draw the Graphics given. Override it and you can put different things.

-g.drawString() draws a String right on the component at the specified location (x&y from top-left).

-JComponent has an automatically executing paintComponent() method to redraw/update the contents. Try not to call it yourself and use repaint() to call it asap.

-Java would rather have Graphics3D be done by external libraries.

-Rectangle2D has a float and double variants because float needs typecasts and it's annoying.

-Ellipse2D has a bounding rectangle as it inherits from RectangularShape

-contains() in Shape class is how you can check for Shapes overlapping each other, like a trigger.

11/18:

-The window retains everything drawn. So to do animation, we have to clear the old graphics somehow. A way is to draw everything again with the new thing moved. That's poor optimization, so another is to draw the same thing again but in the background color.

-Cast the g in paintComponent(Graphics g) to get the specific type of Graphics subclass you want.

-When nothing is happening, the window doesn't re-render itself. You need some action to make it do so, like resizing the window.

-To display an image, use Image or subclasses like ImageIcon to hand to g in paintComponent().

11/22:

-The UML digram-making exercise reminded me how abstract the inheritance structure is. I use JFrame and JPanel all the time but forgot they extend from Component and whatnot.

-JVM constantly monitors events and adds them into a queue to be processed by corresponding methods. Adding too many events to the queue will make the program unresponsive to new events as it is trying to process old ones.

-The event queue ignores events that are unhandled by a method, unlike with exceptions.

?: Exception queue?

-Checked VS Unchecked (Runtime) Exception

11/29: Concurrency, Multitasking VS Multithreading

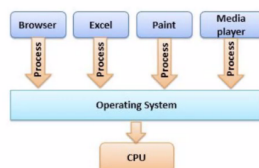
-Multitasking

-The computer (OS) devotes a small piece of time to calculate an instruction(s) of a program, On interrupt, it switched to the next process

-The OS stores the processes in need to calculate in a queue.

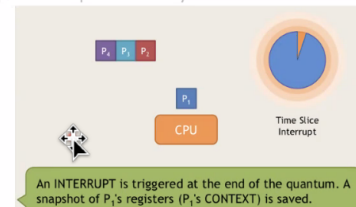
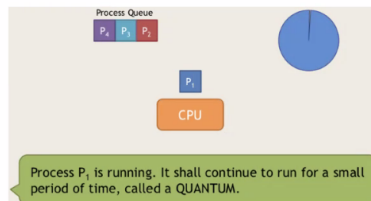
Operating System's ability (seemingly) to run more than one program at a time.

* One CPU or limited number of CPUs (or single-core in a multi-core processor) vs. a large number concurrently executing processes.



* The operating system assigns CPU time slices to each process, giving the impression of parallel activity.

* The operating system assigns CPU time slices to each process, giving the impression of parallel activity.



-Multithreading

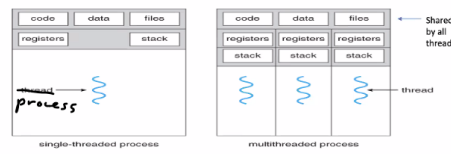
-Executing processes in parallel. You can create, destroy, and assign the work for the thread.

-Threads run different processes on the same data. So, the order in which they run is important to get the right result.

-Java tells the OS that it is using some files, making it inaccessible to edit by other processes.

- Extends the idea of multitasking by taking it **one level lower**:
 - Individual programs/processes will appear to do **multiple tasks** at the same time.
 - Each task is executed in a **thread**, which is short for thread of control.
- Programs that can run one thread at once are said to be **multithreaded**.
- Extremely useful in practice.
 - E.g., a browser can simultaneously download multiple images, a GUI application can have a separate thread for gathering events.

- Each process has a complete set of its own variables while threads share the same data.



java.lang.Thread 1.0
<ul style="list-style-type: none"> • Thread(Runnable target) constructs a new thread that calls the <code>run()</code> method of the specified target. • void start() starts this thread, causing the <code>run()</code> method to be called. This method will return immediately. The new thread runs concurrently. • void run() calls the <code>run</code> method of the associated <code>Runnable</code>. • static void sleep(long millis) sleeps for the given number of milliseconds.
java.lang.Runnable 1.0
<ul style="list-style-type: none"> • void run() must be overridden and supplied with instructions for the task that you want to have executed.

-Ex Using threads in bank accounts

Let's move money between bank accounts:

Bank class stores the balances of a given number of accounts.

transfer() transfers an amount from one account to another.

Thread 1: Moves money from **account 0** to **account 1**.

Thread 2: Moves money from **account 2** to **account 3**.

-Java create and run thread

- 1) Place the code into the **run** method of a class that implements the **Runnable** interface:

```
public interface Runnable {
    void run();
}
Runnable r = () -> { task code }
```

- 2) Construct a **Thread** object from the **Runnable**:

```
var t = new Thread(r);
```

- 3) Start the thread:

```
t.start();
```

-Override the run method of Runnable to execute code.

```
Runnable r = () -> {
    try {
        for (int i = 0; i < STEPS; i++) {
            double amount = MAX_AMOUNT * Math.random();
            bank.transfer(0, 1, amount);
            Thread.sleep((int) (DELAY * Math.random()));
        }
    } catch (InterruptedException e) {
        ...
    }
}

var t = new Thread(r);
t.start();
```

run() method exits when exception occurs

Note that we do not call run() method But start() method of the thread.

-Thread State

-When you create a state, it is in the "runnable" state.

-Then there are other states when stuff happens

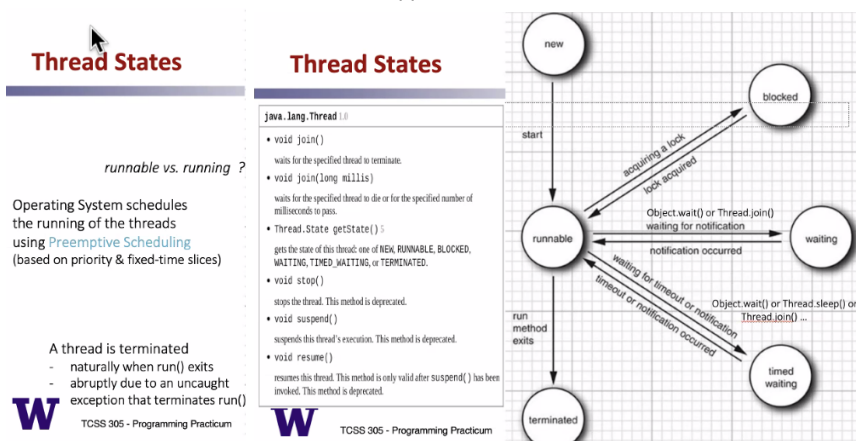


Figure 12.1

- To restart a thread in java
 - Stop it by
 - join() so it forcefully stops (BEST if necessary)
 - Have the run() method finish executing (BEST)
 - wait() sleep()
 - Start it again
 - By making a new one (BEST)
 - Stop it from waiting/sleeping

12/3:

- Daemon thread
 - Makes it run in the background.
- Set Priority
 - Tells JVM or OS idk that a thread should be important/unimportant and the scheduler tries to devote more/less time/turns to.
- If threads don't sleep
 - The scheduler may rapidly do one thread before moving on.
 - You can't trust the ordering of thread execution, so watch out if then use the same data
- Locking
 - You can lock a thread to a block of code, not allow another thread to use it for that time.
- Synchronized
 - Similar to lock, only one thread can run a method at a time.
 - To specify the exact spot
 - use wait() in the so other threads wait if they want to use it.
 - use notifyAll() to stop locking that block of code.

12/8:

- Enum & Enumeration Classes
 - final static...
 - Initialize
 - final static int BRUH1 = 1;
 - final static int BRUH2 = 2;
 - enum BRUHS = {BRUH1, BRUH2, ...}
 - Constructor Bruh(int i) that returns the enum.
 - Becomes a datatype
 - Dont need to .equals() enums because it is a static value.