

What is this?
Why do we care?

One Name

what is stack used for?

-> keep track of functions

-> local variables on stack

-> arguments, return values on stack

- Use **this** to reference the current object

- All objects are accessed by references. objects are on heap

- References are like pointers but
Java automatically dereferences when needed.

- Give each idea one name

- Name field and constructor parameters the same.

- Ex: name both numStudents, vs using each of:

- 1 - studentCount

- 2 - numStudents

- n

- numberStds

```
public class Course {  
    private int numStudents;  
    public Course(int numStudents) {  
        this.numStudents = numStudents;  
    }  
}
```

Pass by value

suppose we have a reference (calling code) to a clock
-> can change properties of the clock
if create a new clock

-> no way to change calling code pointing to new clock from old clock just by passing value

or pointers

reference

- **Java** uses **pass by value**
 - Passing a primitive type passes its value.
 - Passing an object passes (by value) **a reference to the object**
- What this means
 - When passed a primitive type, changes inside a method have no effect outside the method.
 - When passed an object, you *can* modify its state.
 - You *cannot* change **which object it points to**

```
public class Clock {  
    private String background;  
    public setBackground(String background){  
        this.background = background;  
    }  
}
```

not constructor but factory method

```
public Clock makeGreatClock(){  
    return new Clock();  
}
```

```
public void changeClock(){  
    clock.setBackGround("Blue");  
}
```

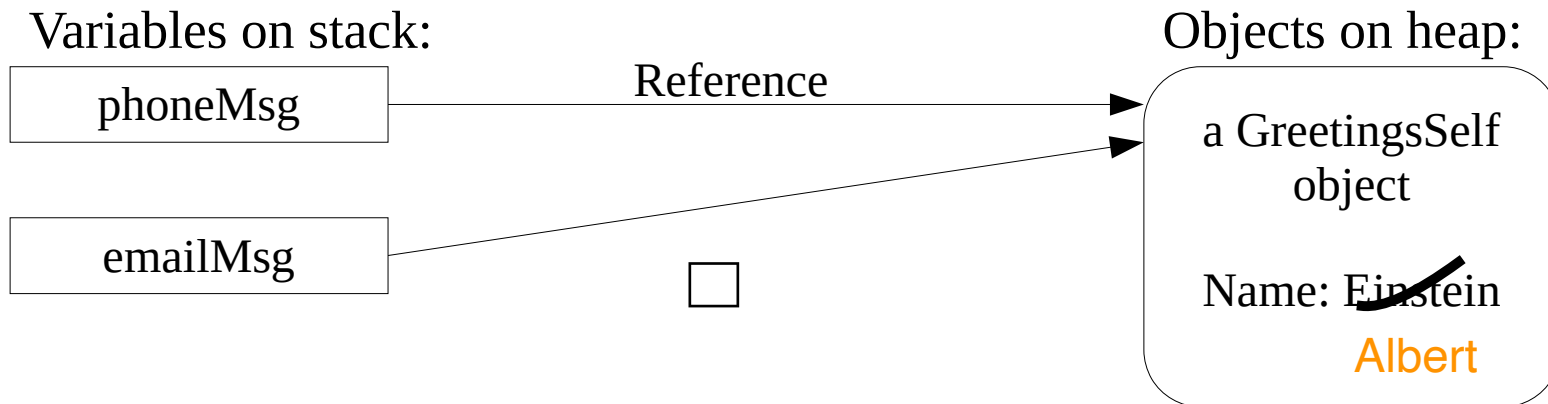
```
public void someCallingCode(){  
    Clock myClock = makeGreatClock();  
    changeClock(myClock);  
}
```

Multiple Object Reference

- = on an **object reference**. copies the reference

- **Example** no operator override in java
copy: shallow vs deep

```
GreetingsSelf phoneMsg = new GreetingsSelf("Einstein");  
GreetingsSelf emailMsg = phoneMsg;  
  
emailMsg.setName("Albert");
```



- **Automatic Garbage Collection**
 - Objects with no references to them are automatically deleted.

Comments

- JavaDoc:
commenting syntax used to generate documentation.
 - on a class: above a class to describe purpose of class
 - on a method: above a method (or field) to explain it
 - Suggest only using for API methods:
stable interface and requires solid documentation
for external users.
- Commenting Rules (this course):
 - RULE 1: comment purpose of each class
 - RULE 2: Name fields, methods, and parameters well so
you don't have to comment them

Integrated Debugger

The screenshot shows an IDE window titled "01-IntroJava_Base - HelloWorld.java". The code editor displays the following Java code:

```
6 public class HelloWorld {  
7     public static void main(String[] args) {  
8         String courseName = "CMPT213";  
9         System.out.println("Hello " + courseName + " World!");  
10    }  
11    }  
12  
13  
14    private static void displayDisclaimer(String courseName) {  
15        System.out.println();  
16        System.out.println("-----");  
17        System.out.println("Legal notice:");  
18        System.out.println("-----");  
19    }  
20 }
```

A breakpoint is set on line 15, indicated by a speech bubble labeled "1. Set breakpoint". The "Debug" toolbar at the top right contains a play button, a speech bubble labeled "2. Run debug", and other debugging icons. The "Debug" panel at the bottom shows the "Frames" tab with the following stack:

- "main"@... RUNNING
- displayDisclaimer:15, HelloWorld
- main:11, HelloWorld

The "Variables" tab shows the variable `courseName` with the value `"CMPT213"`. A speech bubble labeled "4: Step program
F7: Step Into
F8: Step Over
F9: Resume" points to the "Variables" tab. The "Watches" tab is empty, showing "No watches". The "Run" button in the bottom toolbar is highlighted, with a speech bubble labeled "3. Use debugger" pointing to it. The status bar at the bottom indicates "All files are up-to-date (a minute ago)" and "15:1 CRLF UTF-8 Tab* master".

What is the most over-used key word in C-based languages?

Static!

in C++, 4 ways to use static
in Java, 3 ways

Static

when to use static?

—> when you want objects to share/use the same data

- Static method
 - Can be called on the class ^{can be called before creating new object} (no object required).
 - Also called **class methods**
- Static field ^{accessible by all objects of the class}
 - Shared by all instances of the class.
 - Also called **class data**
 - Often used for constants:
`public static final int DAYS_PER_WEEK = 7;`
- Static local ^{in C/C++}
 - Not supported in Java.

Static: What fails to compile?

```
public class StaticFun {  
    public static final int TARGET_NUM_HATS = 10;  
    private static int countNumMade = 0;  
    private int favNum = 0;  
    public static void main(String[] args) {  
        // WHICH OF THESE 4 LINES GIVES A COMPILE TIME ERROR?  
        changeFavNum(42); FAILED - a static function has to call a static function/var  
        displayInfo();  
        favNum = 10; FAILED - same reason  
        countNumMade = 9;  
    }  
    private void changeFavNum(int i) {  
        favNum = TARGET_NUM_HATS + i;  
        displayInfo();  
    }  
    private static void displayInfo() {  
        System.out.println("TARGET_NUM_HATS: " + TARGET_NUM_HATS);  
        System.out.println("countNumMade: " + countNumMade);  
        System.out.println("favNum: " + favNum);  
    }  
}
```

NOTE: constructor needs to create a full form object

Static Factory Method

a method that makes an object, method is static

- Static Factory Method
 - A `static` method which creates an object
 - Like a constructor, but more flexible:
can give it a `descriptive` name
 - A common `design` pattern
- Example
 - In Pizza class:

```
public static Pizza makePizzaFromFile(File file) {  
    // Open file and read in values  
    // Create new Pizza object  
    // Return the Pizza  
}
```

Classes, Strings, Collections,

toString()

- All Java objects have a toString() method
 - All classes inherit from Object, which implements toString()
- Returns a String object which describes the object
 - Used for **debugging**, not formatted screen/file output
 - Recommended format:

```
@Override
public String toString() {
    return getClass().getName()
        + "[daField1=" + daField1
        + ", daField2=" + daField2 + "];"
}
```

@Override Annotation:
method overrides a
base class's method.
(optional)

getClass().getName() returns
class name of current object.

..|
why do we use annotation instead of comment

String Demo

```
static void demoStringConcat() {  
    String guess1 = "hello " + 42;  
    String guess2 = "hello " + 4 + 2;  
    String guess3 = 42 + "hello";  
    String guess4 = 4 + 2 + "hello";  
    String guess5 = new Integer(42).toString();  
}
```

What does each String hold?

Integer, Double, Boolean, Long are Wrappers/Wrapper classes

```
static void demoStringToNumber() {  
    String myInput = "42";  
    int theValue = Integer.parseInt(myInput);  
    // their functions (.parseInt) are static  
  
    // Current date/time to string  
    Date now = new Date();  
    String msg = "Currently " + now;  
    System.out.println(msg);  
  
    // Demo bad conversion  
    int oops = Integer.parseInt("Oops");  
}
```

Also have:

Double.parseDouble(...)
Boolean.parseBoolean(...)
Long.parseLong(...)

Date.toString() gives:

Thu Jan 16 13:49:46 PST 2014

Date in java.util.Date

Throws
NumberFormatException

= DemoStrings.java

Immutable objects

- Strings are Immutable
Once created, they cannot be changed at all
 - To “change” a string, create a new one

- Example

```
String msg = "H";  
msg = msg + "i";  
msg += '!';  
int count = msg.length();
```

Creates 3 strings;
2 for garbage collection:..
“H” and “Hi”

- Java does not support overloaded operators in general, except for + and += on Strings.
 - String still immutable, even with +=

how does it work with +=

Comparing Strings

- Compare strings using `equals()`

```
String password = getDaUsersPassword();  
if (password.equals("12345")) {  
    System.out.println("The air-shield opens.");  
}
```

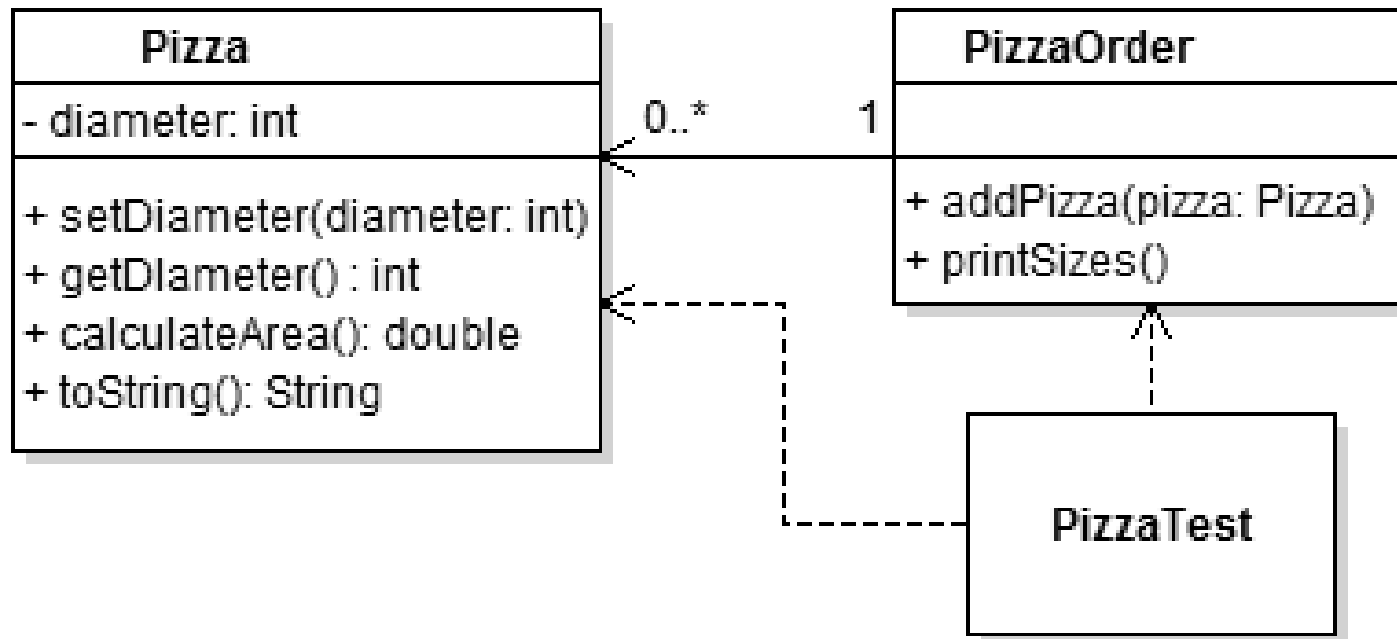
- Don't use `==`

- `==` compares the `memory locations`

```
if (password == yourGuess) {  
    String msg = "Wow! The program stores the "  
        + "password and your guess at the same "  
        + "memory location! Crazy!";  
    System.out.println(msg);  
}
```


UML

- We will create the following classes in this section of the slides.



List and ArrayList

- **Generic**: works with **different types of objects**
- Java includes many generic Collections.
 - ArrayList implements the List **interface** and is backed by an array (fast), and dynamically resizes.

```
List<Hat> hats = new ArrayList<>();  
hats.add(new Hat("Blue"));  
for(Hat hat: hats) {  
    ...  
}
```

Don't need to put <Hat>, the type, because already specified on left-side.

- Collections... **only store objects not primitive types**
 - To store primitives, use built in **wrapper classes**: Integer, Long, Double, etc.
- Why List and ArrayList?
 - Design Principle: **program to an interface, not an implementation**

When is your code done?

Coding Standards

Clean Code

- Correct Code
 - Implements the requirements.
 - Has no (few) bugs.
- Clean Code
 - is correct
 - Conforms to coding standard
 - easy to read
 - easy to maintain
- Professionals write clean code.

Coding Standard

- Course (and most companies) has a coding standard
(See web page)
 - Your code *must* conform to this style guide.
 - Each assignment may mention some specifics.
 - Different than textbook:
 - K&R style bracket placement
 - Always include { }, even on one-line if/else
 - List fields before methods
- Activity
 - Read Coding Standard.
 - Go through the Person class and clean it up.

Summary

- Use one clear name for an idea.
- References to objects, everything pass-by-value.
- Static makes class methods and class data.
- String: Immutable class for working with all strings.
- Show classes with UML class diagram.
- Coding standard enforced for clean code.