

What is this?  
Why do we care?

# One Name

- Use this to..
  - All objects are accessed by **references**.
  - 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:
    - **studentCount**
    - **numStudents**
    - **n**
    - **numberStds**

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

# Pass by value

- **Java uses pass by value**
  - Passing a **primitive** type passes its value.
  - Passing an **object** passes (by value)..
- **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..

# Multiple Object Reference

- = on an object reference..
- Example

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

Variables on stack:

phoneMsg

emailMsg

Reference

Objects on heap:

a GreetingsSelf  
object

Name: Einstein

- 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:..
  - RULE 2: Name fields, methods, and parameters well so
  - ..

# 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) { courseName: "CMPT213"  
15         System.out.println();  
16         System.out.println("-----");  
17         System.out.println("Legal notice:");  
18         System.out.println("-----");  
19     }  
20 }
```

A red breakpoint icon is placed on line 15. A yellow callout bubble points to this icon with the text "1. Set breakpoint".

The top toolbar contains a green play button with a bug icon, which is highlighted by a yellow callout bubble with the text "2. Run debug".

The bottom toolbar contains several debugging icons: a green play button, a red stop button, a blue step into button, a blue step over button, and a blue resume button. A yellow callout bubble points to these icons with the text "3. Use debugger".

The bottom-right pane shows the "Debug" window with the following content:

- Debugger: HelloWorld
- Debugger toolbar: Run, Stop, Step Into, Step Over, Resume, etc.
- Frames: "main"@... RUNNING, displayDisclaimer:15, HelloWorld, main:11, HelloWorld
- Variables: courseName = "CMPT213"
- Watches: No watches.

A yellow callout bubble points to the "Step Into", "Step Over", and "Resume" buttons with the text "4: Step program F7: Step Into F8: Step Over F9: Resume".

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

**Static!**



# Static

- **Static method**
  - Can be called on the class (no object required).
  - Also called..
- **Static field**
  - Shared by all instances of the class.
  - Also called..
  - Often used for constants:  

```
public static final int DAYS_PER_WEEK = 7;
```
- **Static local**
  - 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);
        displayInfo();
        favNum = 10;
        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);
    }
}
```

# Static Factory Method

- **Static Factory Method**

- A..
- Like a constructor, but more flexible:  
can give it a..
- A common..

- **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..
  - Used for **debugging**,..
  - 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.

..

# 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();  
}
```

```
static void demoStringToNumber() {  
    String myInput = "42";  
    int theValue = Integer.parseInt(myInput);  
}
```

```
// 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");  
}
```

What does each String hold?

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

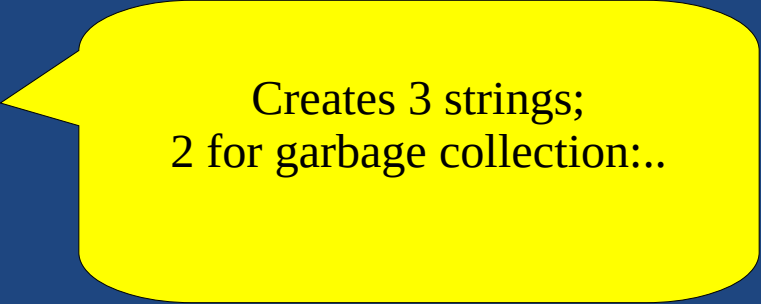
14

# Immutable

- **Strings are Immutable**  
Once created,..
  - To “change” a string,..

- **Example**

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



Creates 3 strings;  
2 for garbage collection:..

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

# Comparing Strings

- Compare strings using..

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

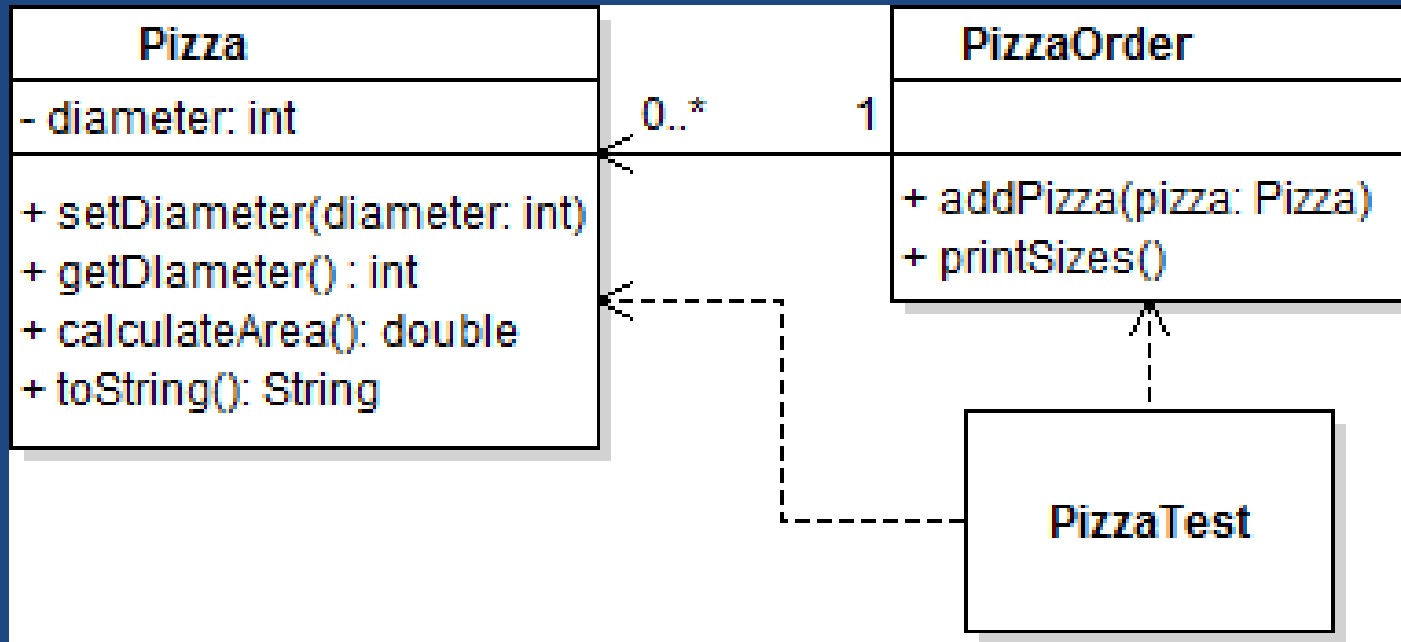
- Don't use ==

- == compares the..

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



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



# List and ArrayList

- **Generic:** works with..
- 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...
  - To store primitives, use built in.. Integer, Long, Double, etc.
- **Why List and ArrayList?**
  - **Design Principle:** ..

When is your code done?  
Coding Standards

# Clean Code

- **Correct Code**
  - Implements the requirements.
  - Has no (few) bugs.
- **Clean Code**
  - 
  - Conforms to..
  - 
  -
- 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**.