6.S092 Introduction to Java

Lecture 3
Outline

- Review methods, Assignment 2
- Popular Issues
- Object Oriented Programming
  - Fields
  - Methods
  - Constructors
  - Composition
  - Objects vs. primitives
- Modifiers
- Multi-class projects
class MyMethods {

    public static void main (String arguments) {

    }

}
class MyMethods {

public static void main (String arguments) {

} // What will you see in Eclipse?
class MyMethods {

/**
 * @param args
 */

public static void main (String[] arguments) {
   // TODO Auto-generated method stub
}
}
Review: Methods

class MyMethods {

    public static void main (String[] arguments) {

    }

}
Review: Methods

class MyMethods {

    public static void main (String[] arguments) {
        // print "Hello World"
    }
}
}
Review: Methods

class MyMethods {

    public static void main (String[] arguments) {
        System.out.println("Hello world!");
    }
}
}
Review: Methods

class MyMethods {

    // make a method called printGreeting that prints "Hello world"

    public static void main (String[] arguments) {
        System.out.println("Hello world!" ComboBox);
    }
}
class MyMethods {

    public static void printGreeting() {
        System.out.println("Hello world!");
    }

    public static void main (String[] arguments) {
        System.out.println("Hello world!")
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting() {
        System.out.println("Hello world!");
    }

    public static void main (String[] arguments) {
        System.out.println("Hello world!") // replace
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting() {
        System.out.println("Hello world!");
    }

    public static void main (String[] arguments) {
        printGreeting();
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting() {
        System.out.println("Hello world!");
    } // Change to specify number of prints

    public static void main (String[] arguments) {
        printGreeting();
    }

}
class MyMethods {

    public static void printGreeting(int num) {
        System.out.println("Hello world!");
    }

    public static void main (String[] arguments) {
        printGreeting();
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting(int num) {
        System.out.println("Hello world!"); //print x num
    }

    public static void main (String[] arguments) {
        printGreeting();
    }
}

class MyMethods {

    public static void printGreeting(int num) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!");
        }
    }

    public static void main (String[] arguments) {
        printGreeting();
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting(int num) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!");
        }
    }

    public static void main (String[] arguments) {
        printGreeting(); // call 3 times
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting(int num) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!");
        }
    }

}

public static void main (String[] arguments) {
    printGreeting(3);
}
}
class MyMethods {

    public static void printGreeting(int num) { // add a param for name
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!");
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3);
    }
}
class MyMethods {

    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!");
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3);
    }
}
Review: Methods

class MyMethods {
    
    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello world!"); // say hello by name
        }
    }
    
    public static void main (String[] arguments) {
        printGreeting(3);
    }
}
class MyMethods {

    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello " + name);
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3);
    }
}
class MyMethods {

    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello " + name);
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3); // call method with updated params
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello " + name);
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3, "Kasia");
    }
}
Review: Methods

class MyMethods {

    public static void printGreeting(int num, String name) {
        for (int i = 0; i < num; i++) {
            System.out.println("Hello "+ name);
        }
    }

    public static void main (String[] arguments) {
        printGreeting(3, "Kasia");
    }

} // Quick tip: Overloading methods
Review: Methods

class MyMethods {

    public static void printGreeting() { ... }
    public static void printGreeting(int num) { ... }
    public static void printGreeting(int num, String name) { ... }

    public static void main (String[] arguments) {
        printGreeting();
        printGreeting(3);
        printGreeting(3, "Kasia");
    }
} // Quick tip: Overloading methods
Review: Method type

```java
public static TYPE NAME() {
    ...
    return EXPRESSION_MATCHING_TYPE;
}
```
Review: Method type

```java
def public static TYPE NAME() {
    ...
    return EXPRESSION_MATCHING_TYPE;
}
```

- types are int, double, boolean, String, etc.
Review: Method type

```java
public static TYPE NAME() {
    ...
    return EXPRESSION_MATCHING_TYPE_TYPE;
}
```

- types are int, double, boolean, String, etc.
- must return same type (not diff type or nothing)
Review: Method type

```java
public static TYPE NAME() {
    ...
    return EXPRESSION_MATCHING_TYPE;
}
```

- types are int, double, boolean, String, etc.
- must return same type (not diff type or nothing)
- `void` means "no type", nothing is returned
Review: Method type

When do you use `void` and when do you return a value?
Review: Method type

When do you use `void` and when do you return a value?

- `void` is for side effects: modify a variable's state, go perform a task like reading/writing to a file, print to the console
Review: Method type

When do you use `void` and when do you return a value?

- **`void`** is for side effects: modify a variable's state, go perform a task like reading/writing to a file, print to the console
- return a value if you need to know or use the product of that method: give back a sum, report whether true or false, return an array you're going to keep working with
Review: Method type

Q: Can you return more than one return value?
Review: Method type

Q: Can you return more than one return value?
A: No. Only one return value and of the same type as specified.
Review: Method type

Q: Can you return more than one return value?
A: No. Only one return value and of the same type as specified.

Q: What return type does printing a String require?
Review: Method type

Q: Can you return more than one return value?
A: No. Only one return value and of the same type as specified.

Q: What return type does printing a String require?
A: Trick qst. Printing to the console is not an instance of returning a value, so printing is not limited to a specific return type.
Review: Method type

class MyMethods {

    public static void printSomething() {
        System.out.println("Something");
    }

    public static String printAString() {
        return "a string";
    }

    public static void main (String[] arguments) {
        ... }

} // Don't mistake returning a String for printing it
Solution to Assignment 2

High level outline:

● Create two separate methods to determine the fastest and second fastest runners
Solution to Assignment 2

High level outline:

- Create two separate methods to determine the fastest and second fastest runners
- Both take in the array of times as a parameter
Solution to Assignment 2

High level outline:

- Create two separate methods to determine the fastest and second fastest runners
- Both take in the array of times as a parameter
- The first method finds the fastest time
Solution to Assignment 2

High level outline:

- Create two separate methods to determine the fastest and second fastest runners
- Both take in the array of times as a parameter
- The first method finds the fastest time
- The second method calls the first to get the fastest time, then finds the fastest time excluding that
Solution to Assignment 2

First method outline:
Solution to Assignment 2

First method outline:

- Declare a `minValue` and `minIndex` variable
Solution to Assignment 2

First method outline:

- Declare a `minValue` and `minIndex` variable
- Create a for loop to iterate through the time values
Solution to Assignment 2

First method outline:

- Declare a `minValue` and `minIndex` variable
- Create a for loop to iterate through the time values
- If the time value you're currently looking at in the loop is faster than the value saved in `minValue`, update `minValue` and `minIndex` with the current value
Solution to Assignment 2

First method outline:

- Declare a `minValue` and `minIndex` variable
- Create a for loop to iterate through the time values
- If the time value you're currently looking at in the loop is faster than the value saved in `minValue`, update `minValue` and `minIndex` with the current value
- Return `minIndex`
public static int getMinIndex(int[] values) {
    int minValue = Integer.MAX_VALUE;
    int minIndex = -1;
    for (int i = 0; i < values.length; i++) {
        if (values[i] < minValue) {
            minValue = values[i];
            minIndex = i;
        }
    }
    return minIndex;
}
Solution to Assignment 2

Second method outline:
Solution to Assignment 2

Second method outline:

- Declare a `secondIdx` and `minIdx` variable
Solution to Assignment 2

Second method outline:

- Declare a `secondIdx` and `minIdx` variable
- Create a for loop to iterate through the time values
Solution to Assignment 2

Second method outline:

- Declare a `secondIdx` and `minIdx` variable
- Create a for loop to iterate through the time values
- If the time value you're currently looking at in the loop equals `minIdx`, continue
Solution to Assignment 2

Second method outline:

- Declare a `secondIdx` and `minIdx` variable
- Create a for loop to iterate through the time values
- If the time value you're currently looking at in the loop equals `minIdx`, continue
- Else, if it's less than the value at `secondIdx`, update `secondIdx`
Solution to Assignment 2

Second method outline:

- Declare a `secondIdx` and `minIdx` variable
- Create a for loop to iterate through the time values
- If the time value you're currently looking at in the loop equals `minIdx`, continue
- Else, if it's less than the value at `secondIdx`, update `secondIdx`
- Return `secondIdx`
public static int getSecondMinIndex(int[] values) {
    int secondIdx = -1;
    int minIdx = getMinIndex(values);
    for (int i = 0; i < values.length; i++) {
        if (i == minIdx)
            continue;
        if (secondIdx == -1 || values[i] < values[secondIdx])
            secondIdx = i;
    }
    return secondIdx;
}
Solution to Assignment 2

Main method outline:
Solution to Assignment 2

Main method outline:

- Declare `minIndex` and `secondIdx` and instantiate them to the values returned by calling `getMinIndex` and `getSecondMinIndex`, respectively
Solution to Assignment 2

Main method outline:

- Declare `minIndex` and `secondIdx` and instantiate them to the values returned by calling `getMinIndex` and `getSecondMinIndex`, respectively.
- Access values in the names and times arrays, using `array_name[index]` format.
Solution to Assignment 2

Main method outline:

- Declare `minIndex` and `secondIdx` and instantiate them to the values returned by calling `getMinIndex` and `getSecondMinIndex`, respectively.
- Access values in the names and times arrays, using `array_name[index]` format.
- Print the first and second fastest runners and their times.
Solution to Assignment 2

```java
public static void main(String[] arguments) {
    String[] names = { ... };
    int[] times = { ... };

    int minIndex = getMinIndex(times);
    int secondIdx = getSecondMinIndex(times);

    System.out.println("Fastest: " + names[minIndex] + ", " + times[minIndex]);
    System.out.println("Second fastest " + names[secondIdx] + ", " + times[secondIdx]);
}
```
Solution to sleepIn

- Return `true` if it is not a weekday or we're on vacation.
Solution to sleepIn

- Return **true** if it is not a weekday or we're on vacation.

```java
public static boolean sleepIn(boolean weekday, boolean vacation) {
    return (!weekday || vacation);
}
```
Solution to sleepIn

- Return **true** if it is not a weekday or we're on vacation.

```java
public static boolean sleepIn(boolean weekday, boolean vacation) {
    // return (!weekday || vacation);
    if (weekday == false || vacation == true) return true;
    return false;
}
```
Solution to sleepIn

● Return true if it is not a weekday or we're on vacation.

```java
public static boolean sleepIn(boolean weekday, boolean vacation) {
    // return (!weekday || vacation);
    /* if (weekday == false || vacation == true) return true;
    return false; */
    if (weekday == false) return true;
    else if (vacation == true) return true;
    else return false;
}
```
Solution to makeOutWord

- Return a new string where the word is in the middle of the out string, e.g. "<<word>>".
Solution to makeOutWord

- Return a new string where the word is in the middle of the out string, e.g. "<<word>>".

```java
public static boolean makeOutWord(String out, String word) {
    return out.substring(0, 2) + word + out.substring(2);
}
```
Solution to makeOutWord

- Return a new string where the word is in the middle of the out string, e.g. "<<word>>".

```java
public static boolean makeOutWord(String out, String word) {
    // return out.substring(0, 2) + word + out.substring(2);
    String beginning = out.substring(0, 2);
    String ending = out.substring(2, 4);
    String result = beginning + word + ending;
    return result;
}
```
CodingBat background

What's happening in the background?
CodingBat background

What's happening in the background?

● Your method is called with a variety of parameters to test that it works for the range of possible inputs:
CodingBat background

What's happening in the background?

- Your method is called with a variety of parameters to test that it works for the range of possible inputs:

```java
if (sleepIn(false, false) == false) { // Should return true
    System.out.println("Test 1 Error: SleepIn");
}
if (sleepIn(true, false) == true) { // Should return false
    System.out.println("Test 2 Error: SleepIn");
}
```
CodingBat background

What's happening in the background?

- You see the results of the tests that CodingBat ran for you—what was expected and what your code produced:

<table>
<thead>
<tr>
<th>Expected</th>
<th>This Run</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sleepIn(true, false) → true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>sleepIn(true, false) → false</td>
<td>true</td>
<td>X</td>
</tr>
<tr>
<td>sleepIn(false, true) → true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>sleepIn(true, true) → true</td>
<td>false</td>
<td>X</td>
</tr>
</tbody>
</table>
class CodingBat {
    public static boolean sleepIn(boolean weekday, boolean vacation) {
        return (!weekday || vacation);
    }

    public static void main (String[] arguments) {
        if (sleepIn(false, false) == false) {
            System.out.println("Test 1 Error: SleepIn");
        }
    }
}
Popular Issues 1

Defining a method inside a method

```java
public static void main(String[] arguments) {
    public static void foobar() {
    }
}
```
Popular Issues 2

- Array **Index** vs Array **Value**

```java
int[] values = {99, 100, 101};
System.out.println(values[0]); // 99
```

<table>
<thead>
<tr>
<th>Values</th>
<th>99</th>
<th>100</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexes</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Popular Issues 2

- Array **Index** vs Array **Value**

```java
int[] values = {99, 100, 101};
System.out.println(values[0]);    // 99
```

What are two ways you could get the value 100?

<table>
<thead>
<tr>
<th>Values</th>
<th>99</th>
<th>100</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexes</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
File saved as FileName.java

class filename {
    ...
}
}
File saved as FileName.java

class filename {
    ...
}

// Will this compile?
File saved as FileName.java

class filename {
    ...
}

// Will this compile?
// Nope. Change class name or file name to match.
Popular Issues 3

File saved as FileName.java

```java
class FileName {
    ...
}
// Will this compile?
// Yes. This will compile now.
```
Debugging Notes 1

System.out.println is your friend

```java
for ( int i=0; i< vals.length; i++) {
    if ( vals[i] < minVal) {
        System.out.println("cur min: " + minVal);
        System.out.println("new min: " + vals[i]);
        minVal = vals[i];
    }
}
```
Formatting

**Ctrl-shift-f** is your friend

```java
for (int i = 0; i < vals.length; i++) {
    if (vals[i] < vals[minIdx]) {
        minIdx = i;
    }
}
return minIdx;
```

Is there a bug? Who knows! Hard to read
Object Oriented Programming
Object Oriented Programming

- Represent the real world

Baby
Object Oriented Programming

- Represent the real world

<table>
<thead>
<tr>
<th>Baby</th>
<th>Name</th>
<th>Sex</th>
<th>Weight</th>
<th>Decibels</th>
</tr>
</thead>
</table>

Object Oriented Programming

- Objects group together:
  - Primitives (int, double, char, etc.)
  - Objects (String, etc.)

Baby

String name
boolean isMale
double weight
double decibels
Why use **classes**?

- Why not just primitives?

  ```java
  // little baby alex
  String nameAlex;
  double weightAlex;
  // little baby david
  String nameDavid;
  double weightDavid;
  ```
Why use **classes**?

- Why not just primitives?

```java
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
// little baby david
String nameDavid2;
double weightDavid2;
```

David2? Terrible 😞
Why use **classes**?

- Why not just primitives?

```java
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
// little baby david
String nameDavid2;
double weightDavid2;
```

David2? Terrible 😞

500 Babies? That Sucks!
Why use **classes**?

Baby1

- Name
- Weight
- Sex
  ...

![Diagram showing a class with attributes: Name, Weight, Sex, and an instance: Baby1.](image-url)
Why use **classes**?

<table>
<thead>
<tr>
<th>Name</th>
<th>Weight</th>
<th>Sex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baby 1</td>
<td>496</td>
<td></td>
</tr>
<tr>
<td>Baby 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baby 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baby 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Why use classes?

Nursery

Baby 1  Baby 2  Baby 3  Baby 4  496 more Babies …
Why use **classes**?

Nursery

<table>
<thead>
<tr>
<th>Nurse 1</th>
<th>Nurse 2</th>
<th>Nurse 3</th>
<th>Nurse 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baby 1</td>
<td>Baby 2</td>
<td>Baby 3</td>
<td>Baby 4</td>
</tr>
</tbody>
</table>

More nurses…

496 more Babies …
Why use classes?

Nursery

- Nurse
- Baby
Why use **classes**?

Hospital

Nursery

Baby

Nurse
Defining classes
public class Baby {

}
public class Baby { // We start with the class name
}
public class Baby {

} // We proceed to fill in three sections that make up a class
public class Baby {

} // Fields: states of the object altered through methods
public class Baby {

Field

Methods

} // Methods: Alter or return fields, perform other actions
public class Baby {

Field(s)

Methods

} // Constructor (optional) : Fills in the default values for fields
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale){ … }
}
Overview: Class Definition

public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    void sayHi() { ... }
    void eat(double foodWeight) { ... }

    Baby(String myname, boolean bMale) { ... }
}

Fields

Methods

Constructor
public class Baby {
    String name;
    double weight = 5.0; // Instantiate default value
    boolean isMale;

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale) { … }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    // Any type name pair is allowed, not just primitive types

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale) { … }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    // How would you write an array of other baby objects?

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale){ … }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    // How would you write an array of other baby objects?
    // Hint: the type is the class name

    void sayHi() {
    }
    void eat(double foodWeight) {
    }

    Baby(String myname, boolean bMale) {
    }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    Baby[] siblings;
    // How would you write an array of other baby objects?
    // Hint: the type is the class name
    void sayHi() { ... }
    void eat(double foodWeight) { ... }
    Baby(String myname, boolean bMale){ ... }
}"
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale) { … }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    void eat(double foodWeight) {
        weight = weight + foodWeight;
    } // Changes field value

    Baby(String myname, boolean bMale) {
    }
}
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;

    void sayHi() { … }
    void eat(double foodWeight) { … }

    Baby(String myname, boolean bMale){
        name = myname; isMale = bMale;
    } // Sets field values with params
}
More on Constructors

- Constructor name == the class name
- No return type – never returns anything
- Usually initialize fields
- All classes need at least one constructor
  - If you don’t write one, defaults to

```java
CLASSNAME () {
}
```
Using classes
Classes and Instances

// class Definition
public class Baby {...}

// class Instances
Baby bart = new Baby(“Bart Simpson”, true);
Baby lisa = new Baby(“Lisa Simpson”, false);
Classes and Instances

// class Definition
public class Baby {...}
// This is what we just finished in the overview

// class Instances
Baby bart = new Baby("Bart Simpson", true);
Baby lisa = new Baby("Lisa Simpson", false);
Classes and Instances

// class Definition
public class Baby {...}
// This is what we just finished in the overview

// class Instances
Baby bart = new Baby("Bart Simpson", true);
Baby lisa = new Baby("Lisa Simpson", false);

// Remember that our constructor called for a String name and a boolean specifying whether a boy
Accessing fields

- Object.FIELDNAME

```java
Baby bart = new Baby("Bart Simpson", true);
System.out.println(bart.name);
System.out.println(bart.weight);
```
Accessing fields

- Object.<FIELDNAME>

```java
Baby bart = new Baby("Bart Simpson", true);
System.out.println(bart.name); // What does this output?
System.out.println(bart.weight); // And this?
```
Calling Methods

- Object.\texttt{METHODNAME}([\texttt{ARGUMENTS}])

```java
Baby bart = new Baby("Bart Simpson", true);
bart.sayHi(); // "Hi, my name is Bart Simpson"
bart.eat(1); // Adds foodWeight to weight value
```
Multiple classes using Eclipse
1) Create package
1) Create package

- Files > New > Java Project
1) Create package

- Files > New > Java Project
- Project Name: IAP_Java (your preference)
1) Create package

- Files > New > Java Project
- Project Name: IAP_Java (your preference)
- Finish
2) Create class
2) Create class

- File > New > Class
2) Create class

- File > New > Class
- Name: Runner_LastName
2) Create class

- File > New > Class
- Name: Runner_LastName
- (Optional) Check the checkbox for the main method stub
2) Create class

- File > New > Class
- Name: Runner_LastName
- (Optional) Check the checkbox for the main method stub
- Finish
2) Create class

- File > New > Class
- Name: Runner_LastName
- (Optional) Check the checkbox for the main method stub
- Finish

Do the same for Race_LastName
2) Create class

- File > New > Class
- Name: Runner_LastName
- (Optional) Check the checkbox for the main method stub
- Finish

Do the same for Race_LastName
3) Complete assignment

- For each class, replace the auto-gen code with given assignment example framework code
3) Complete assignment

- For each class, replace the auto-gen code with given assignment example framework code
- Make sure the code compiles before you turn it in
3) Complete assignment

- For each class, replace the auto-gen code with given assignment example framework code
- Make sure the code compiles before you turn it in
- Even if it prints nothing it should, it should at least compile
3) Zip files and upload
3) Zip files and upload

- Create a single zip file called *exactly* Assignment3_LastName.zip
3) Zip files and upload

- Create a single zip file called *exactly* Assignment3_LastName.zip
- Within this zip file should be *only* two things: Runner_LastName.java and Race_LastName.java
3) Zip files and upload

- Create a single zip file called *exactly* Assignment3_LastName.zip
- Within this zip file should be *only* two things: Runner_LastName.java and Race_LastName.java
- No subdirectories
3) Zip files and upload

- Create a single zip file called *exactly* Assignment3_LastName.zip
- Within this zip file should be *only* two things: Runner_LastName.java and Race_LastName.java
- No subdirectories
- Upload *once* to Stellar
Primitives versus Objects
Primitives vs Objects

- **Primitive** types are basic Java types
  - int, long, double, boolean, char, short, byte, float
  - The actual **values** are stored in the variable
Primitives vs Objects

- **Primitive** types are basic Java types
  - int, long, double, boolean, char, short, byte, float
  - The actual **values** are stored in the variable

- **Objects** types are not simple values
  - String, int[], Baby, …
How java stores primitives

- Variables are like fixed size cups
- Primitives are small enough that they just fit into the cup
How Java stores objects

- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object
How Java stores objects

- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object
References

- The object’s location is called a reference
References

- The object’s location is called a reference
- `==` compares the references
The object’s location is called a reference

`==` compares the references

```java
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

Q: Does `shiloh1 == shiloh2`?
The object’s location is called a reference

`==` compares the references

```java
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

Q: Does `shiloh1 == shiloh2`?
A: No!
References

```java
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```
References

```java
Baby mybaby = new Baby("davy", true)
mybaby.name = "david"  // We change the name
```

```
name = "davy"
isMale = true
...
```

```
mybaby's location
```

Baby mybaby = new Baby("davy", true)
mybaby.name = "david"

// Does that affect mybaby location?
Baby mybaby = new Baby("davy", true)
mybaby.name = "david"

// Does that affect mybaby location?
// No.
References

- Using `= updates the reference.`
References

- Using `=` updates the reference.

```plaintext
baby1 = baby2;
```
References

- using [ ] or .
  - Follows the reference to the object
  - May modify the object, but never the reference
- Imagine
  - Following directions to a house
  - Moving the furniture around
- Analogous to
  - Following the reference to an object
  - Changing fields in the object
void doSomething(int newX, int[] newYs, Baby newB) {
    x = newX;
    ys[0] = newYs;
    b.name = newB.name;
}

...  
int i = 0;
int[] j = {7};
Baby k = new Baby("Shahar", true);
doSomething(i, j, k);

i=? j=? k=?