public class References
{
    public static void main(String[] args)
    {
        MyClass foo = new MyClass();

        System.out.println(foo.getA());
        changeMyClass(foo);
        System.out.println(foo.getA());
    }
}
Topic 09: References

- What is a reference?
- null
- Aliasing
- Array references
What is a Reference?

- The \textbf{address} of an object
- Shows \textbf{where} the object is stored in memory

- Really, it's an integer
  - Index into a gigantic \texttt{byte[]} array
  - But you cannot cast to/from numeric types!
What Variables are References?

- Object variables
- Array variables
  (including arrays of primitives)

- Anything that isn't a primitive!

  byte    short    int    long
  float   double   char   boolean
Storing by Indirection

String `str` = "Hello world!";
int `i` = 47;
double `x` = 1.3;
int[] `data` = new int[100];

<table>
<thead>
<tr>
<th>str</th>
<th>Location 0x123 in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>47</td>
</tr>
<tr>
<td>x</td>
<td>1.3</td>
</tr>
<tr>
<td>data</td>
<td>0x456</td>
</tr>
</tbody>
</table>

Location 0x123 in memory

Location 0x456 in memory
Storing by Indirection

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Location in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>&quot;Hello world!&quot;</td>
<td>0x123</td>
</tr>
<tr>
<td>i</td>
<td>47</td>
<td>0x456</td>
</tr>
<tr>
<td>x</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>int[100]</td>
<td></td>
</tr>
</tbody>
</table>

Two of these variables are primitives, and so they are literally stored inside the variables.

Location 0x123 in memory
Location 0x456 in memory
Two of these variables are references, and so the variables stored address of the variables.

```
String str = "Hello world";
int i = 47;
double x = 1.3;
int[] data = new int[100];
```
So Why Does it Matter?

- References add immense flexibility & efficiency
  - Resize arrays
  - Pass references to methods
  - null references
  - Aliasing
  - Data structures
  - Garbage collection
Topic 09: References

- What is a reference?
- `null`
- Aliasing
- Array references
Pointing at Nothing

• How to express “no object”? 

• null (that's a Java keyword!)
  - Address 0
  - Never a valid value to use
  - But perfectly OK to store
  - Also OK to use in == and != comparisons
null

String str1 = "Hello world!";
int    len1 = str1.length();
System.out.println(len1);

String str2 = null;
int    len2 = str2.length();
System.out.println(len2);

The second length() call is a problem!
Let's see why.
String str1 = "Hello world!";
int len1 = str1.length();
System.out.println(len1);

String str2 = null;
int len2 = str2.length();
System.out.println(len2);

We create a String object.
We save its address into str1.
String str1 = "Hello world!";
int    len1 = str1.length();
System.out.println(len1);

String str2 = null;
int    len2 = str2.length();
System.out.println(len2);

str1 = 0x789
len1 = 12
str2 = 
len2 =

"Hello world!"
length()

We use the str1 variable to find the length() method inside the object.
String str1 = "Hello world!";
int len1 = str1.length();
System.out.println(len1);

String str2 = null;
int len2 = str2.length();
System.out.println(len2);

str1 = 0x789
len1 = 12
str2 =
len2 =

"Hello world!"
String str1 = "Hello world!";
int len1 = str1.length();
System.out.println(len1);

String str2 = null;
int len2 = str2.length();
System.out.println(len2);

We set str2 to null - that is, "no object."
String str1 = “Hello world!”;
int    len1 = str1.length();
System.out.println(len1);

String str2 = null;
int    len2 = str2.length();
System.out.println(len2);

<table>
<thead>
<tr>
<th>str1</th>
<th>len1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x789</td>
<td>12</td>
</tr>
</tbody>
</table>

We attempt to use str2 to find a length() method inside some object.

But str2 doesn't point to any object!
String str1 = "Hello world!";
int len1 = str1.length();
System.out.println(len1);

String str2 = null;
int len2 = str2.length();
System.out.println(len2);

Java reports this error to us with an exception:

NullPointerException

This exception occurs any time we attempt to use null.
Comparing to null

- It's illegal to ever use a null reference
- It's always OK to compare a reference to null

```java
public static int myMethod(Person p) {
    if (p == null)
        throw new IllegalArgumentException();
    ...
```
Topic 09: References

- What is a reference?
- null
- Aliasing
- Array references
Aliasing

- **Aliasing** means referring to the same object using two difference reference variables.
- Each reference is a fully-functional variable, no limitations
- Changes made using one reference affect what you see with the other

```plaintext
obj1 = 0xabc
obj2 = 0xabc
```
Why Aliasing?

- Avoids copying
- Allows object sharing
- Flexible “ownership”
What's Wrong with Aliasing?

- Prevents copying
  - You can do it by hand, but it takes extra work

- Accidental sharing

- Confused ownership

This is the same list!
This is a tradeoff.
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}

russ = 0x314
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}

russ = 0x314

hasBeard=true
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}

russ = 0x314

inst = 0x314

hasBeard=true
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}

russ = 0x314
inst = 0x314

hasBeard=false
Aliasing and Method Calls

Instructor russ = new Instructor();
russ.hasBeard = true;
foo(russ);
System.out.println(russ.hasBeard);

void foo(Instructor inst)
{
    inst.hasBeard = false;
}

russ = 0x314
Limits of Aliasing

- Changing the **object** affects other aliases
- Changing the **variable** does not

```
Instructor russl1 = new Instructor();
russl1.hasBeard = true;
Instructor russ2 = russl1;
russl1 = new Instructor();
russl1.hasBeard = false;
```
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = 0xdef
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = 0xdef

hasBeard=true
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = 0xdef

russ2 = 0xdef

hasBeard=true
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = 0xf00

russ2 = 0xdef
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = \text{\texttt{0xf00}}

russ2 = \text{\texttt{0xdef}}

\text{\texttt{hasBeard=true}}

\text{\texttt{hasBeard=false}}
Limits of Aliasing

Instructor russ1 = new Instructor();
russ1.hasBeard = true;
Instructor russ2 = russ1;
russ1 = new Instructor();
russ1.hasBeard = false;

russ1 = 0xf00

russ2 = 0xdef

hasBeard=true

hasBeard=false
public static void resizeArray(int[] array, int newSize)
{
    int[] copy = new int[newSize];
    copyLen = Math.min(array.length, newSize);
    for (int i=0; i<copyLen; i++)
    {
        copy[i] = array[i];
    }
    array = copy;
}
Limits of Aliasing

public static void resizeArray(int[] array, int newSize) {
    int[] copy = new int[newSize];
    copyLen = Math.min(array.length, newSize);
    for (int i = 0; i < copyLen; i++) {
        copy[i] = array[i];
    }
}

This creates a new array, which is unrelated to the original.

The caller's array is not affected.

How do we fix this?
Limits of Aliasing

```java
public static int[] resizeArray(int[] array, int newSize)
{
    int[] copy = new int[newSize];

    copyLen = Math.min(array.length, newSize);

    for (int i=0; i<copyLen; i++)
    {
        copy[i] = array[i];
    }

    return copy;
}
```
Topic 09: References

- What is a reference?
- null
- Aliasing
- Array references
Array Variables are References

- Arrays are actually **objects**!
- Array variables are references

- Arrays are allocated on the heap (with `new`)

- Weird: `null` array references are allowed
Array Variables are References

public static void zeroHalf(int[] array) {
    for (int i=0; i<array.length; i++) {
        if (i % 2 == 0) {
            array[i] = 0;
        }
    }
}

In Class:
What does this method do?
Array Variables are References

```java
public static void zeroHalf(int[] array) {
    for (int i=0; i<array.length; i++) {
        if (i % 2 == 0) {
            array[i] = 0;
        }
    }
}
```

This method takes an array as input.
Array Variables are References

public static void zeroHalf(int[] array)
{
    for (int i=0; i<array.length; i++)
    {
        if (i % 2 == 0)
        {
            array[i] = 0;
        }
    }
}

It modifies some of the values in the array.
Array Variables are References

public static void zeroHalf(int[] array) {
    for (int i=0; i<array.length; i++) {
        if (i % 2 == 0) {
            array[i] = 0;
        }
    }
}

It never returns anything – yet the changes will be visible to the caller, since array variables are aliases.
Array Variables Can be Null

What is the difference between the following three declarations?

```java
int[] arr1 = new int[4];
int[] arr2 = new int[0];
int[] arr3 = null;
```
Array Variables Can be Null

```java
int[] arr1 = new int[4];
int[] arr2 = new int[0];
int[] arr3 = null;
```

```
arr1 = 0x123
```

```java
```
Array Variables Can be Null

```java
int[] arr1 = new int[4];
int[] arr2 = new int[0];
int[] arr3 = null;
```

```
arr1 = 0x123
arr2 = 0x456
```
Array Variables Can be Null

```java
int[] arr1 = new int[4];
int[] arr2 = new int[0];
int[] arr3 = null;
```

```
arr1 = 0x123
arr2 = 0x456
arr2 = 0x000
```
Something to Ponder

- You can create an array of any type...
- An array is a type...

- So a 2D array is an array of arrays.
- But how is it implemented?
Topic 09: References

- What is a reference?
- null
- Aliasing
- Array references

Summary