

Primitive data types

- count
- cost
- ‘M’, ‘W’, ‘F’
- true



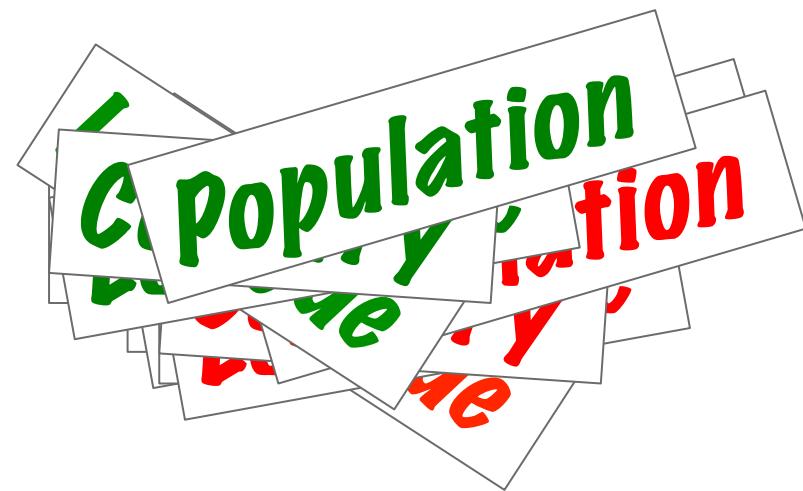
How can we model

- student
- square
- city
- chess board



an array of cities

How can we model



How can we model a city

- Update the city's population
- Calculate the distance between two city's
- ~~Move a rook 3 spaces~~



Object Oriented Programming

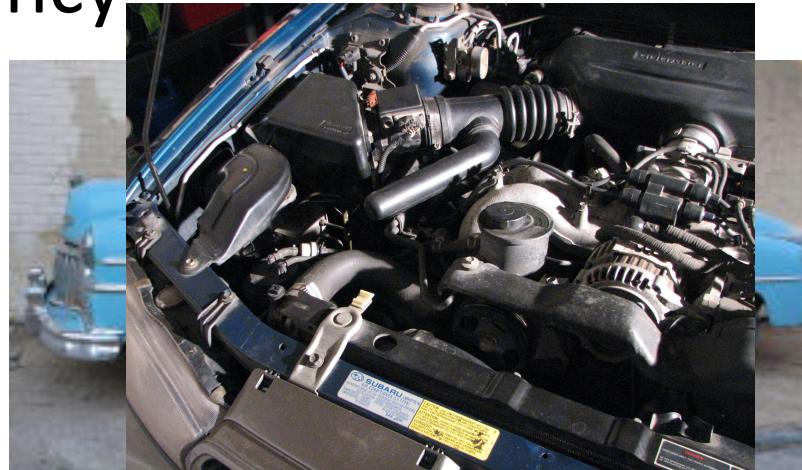
Its terminology

OOP Terminology

- **object:** – Placing behavior in objects.
- **object:** writing programs that perform most of their useful behavior through interactions with objects.A diagram illustrating the structure of an object. It consists of two adjacent rectangular boxes. The left box has a yellow border and contains five items: Name, Lat, Long, Population, and Country. The right box has an orange border and contains two methods: updatePopulation() and distanceFrom(anotherCity). The entire diagram is enclosed in a larger yellow border.

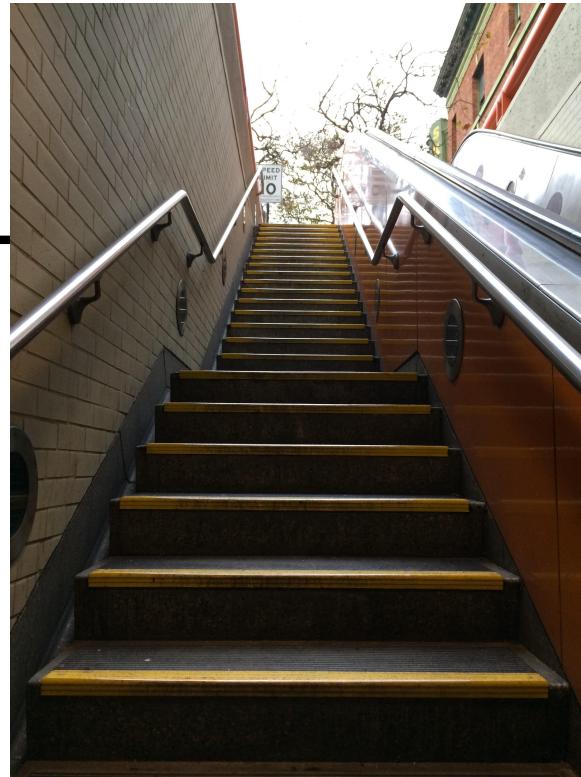
A wonderful side effect to creating and using objects

- **abstraction:** to hide much of the details about an object in order to reduce complexity and increase efficiency



Objects

out.println("Hello world!");



OOP Terminology

A class

- A program or module
 - You have written several class files already, they contain a main module and executable code
- The template for an object
 - City, Student or ChessBoard

An object

- An instance of a class
 - Philadelphia
 - S13792
 - Game3

Overview of Java String class

String class

- A class file that describes the data and methods associated with String objects
 - a sequence of characters
 - methods such as concat, contains, split

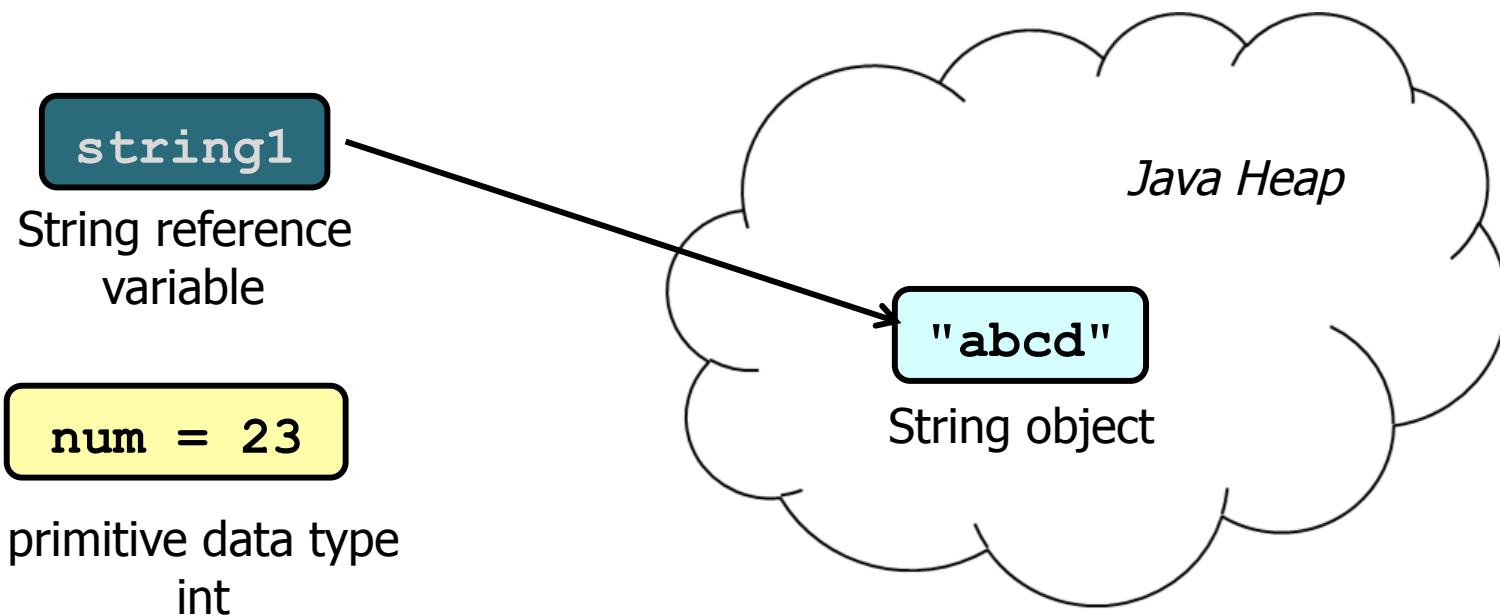
String object

- “apple”
- “Hello World”
- “This is a long String”

Java String class

- String is a class that represents a sequence of characters

```
String string1 = "abcd";
```



Examples where String and primitive data types act in the **same** way

String object

Prints "abcd" String literal

```
out.println("abcd") ;
```

int primitive data type

Prints 23

```
out.println(23) ;
```

We can use + with String objects

String object

Prints "wxyz" String literal

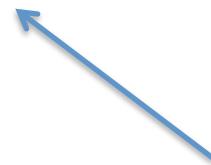
```
out.println("wx" + "yz");
```

int primitive data type

Prints 27

```
out.println(23+4);
```

**Will perform concatenation of the
two String objects**



Here is where the similarities end

*a sequence
of
characters*

methods on String objects

Java String class

Assign String literal "cde" to a variable

```
String str1= "cde";
```

Concatenate & print "cdexyz"

```
out.println(str1.concat("xyz"));
```

Extract substring beginning at a and ending before c

```
String str2 = "abc".substring(0,2);
```

Prints "ab"

```
out.println(str2);
```

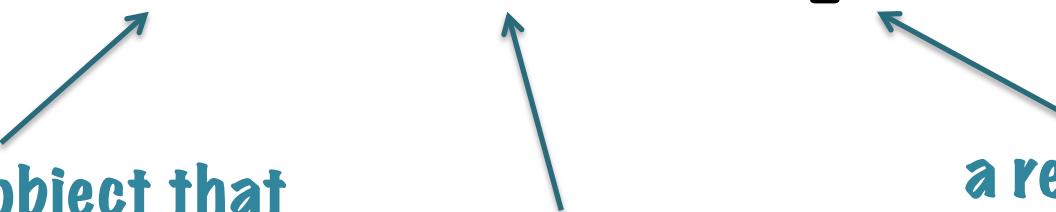
Using methods with objects

```
str1.concat("xyz")
```

the String object that
is calling the method

the name of
the method

a required
parameter



The diagram illustrates the components of a method call. At the top, the code `str1.concat("xyz")` is shown in black. Three arrows point from blue explanatory text below to specific parts of the code: one arrow points to `str1` with the label "the String object that is calling the method"; another arrow points to `concat` with the label "the name of the method"; and a third arrow points to `"xyz"` with the label "a required parameter".

Using methods with objects

```
out.print(str1.concat("xyz"));
```

This method returns a new string that is the concatenation of str1 and "xyz"

Other examples of String methods

- *<String object name>.substr (x ,y) ;*
 - returns the string beginning at character x and ending just before character y
- *<String object name>.startsWith (substr2) ;*
 - returns true if the String begins with the substr2
- *<String object name>.length () ;*
 - returns an integer which is the length of the String object

Comparing two Strings

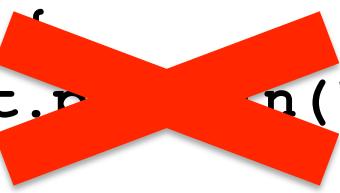
```
String a = "may";  
String b = "May";  
  
if (a == b) {  
    System.out.println("a == b");  
}
```

Never compare objects using ==
This will compile, but the results may
not be what you are really asking for

Comparing two Strings

```
String a = "may";  
String b = "May";
```

```
if (a == b) {  
    System.out.println("a == b");  
}
```



```
if (a.equals(b)) {  
    System.out.println("a.equals(b)");  
}
```

Always use the `equals` method
when comparing two objects

Why is this the case?

- Let's build a class of our own and then we can see what is going on behind the scenes

