# Primitive data types

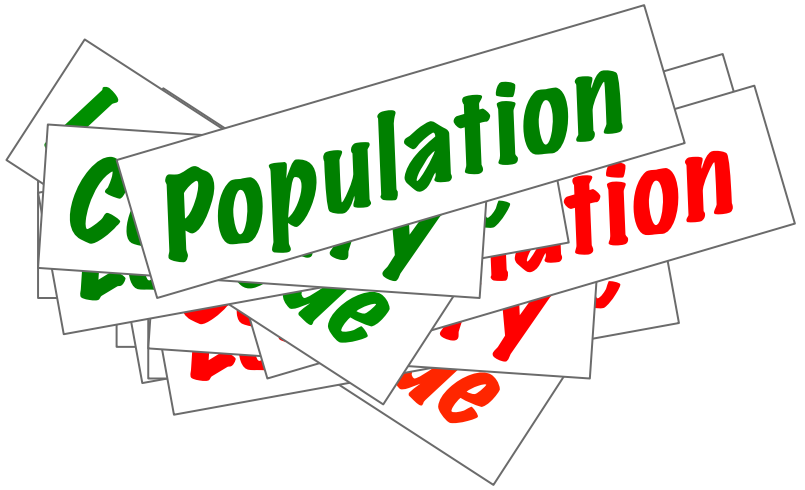
- count
- cost
- ‘M’, ‘W’, ‘F’
- true

# How can we model

- student
- square
- city
- chess board

# How can we model

an array of cities



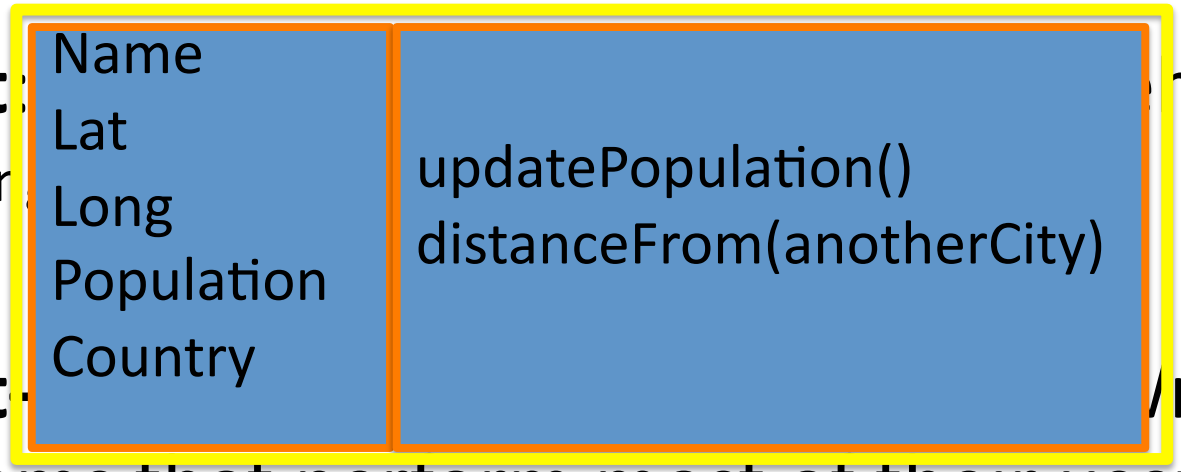
# How can we model a city

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

# Object Oriented Programming

## Its terminology

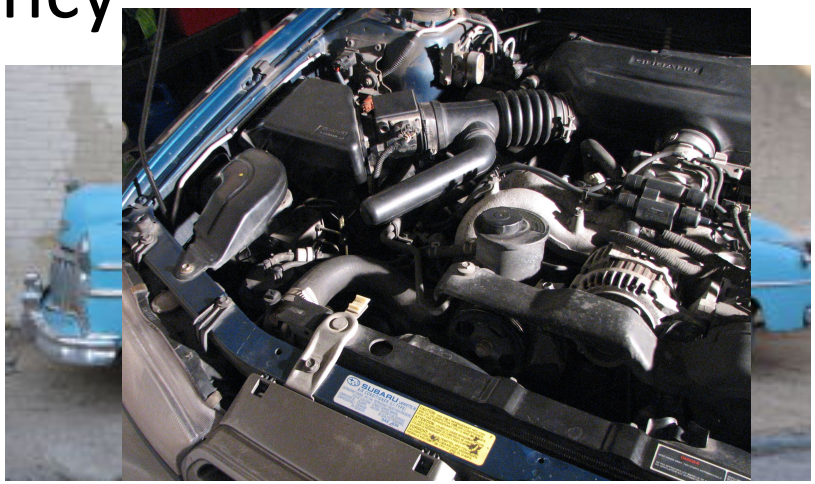
# OOP Terminology




- **object**: …navior.
  - Placir…
- **object**-… …riting programs that perform most of their useful behavior through interactions with objects.

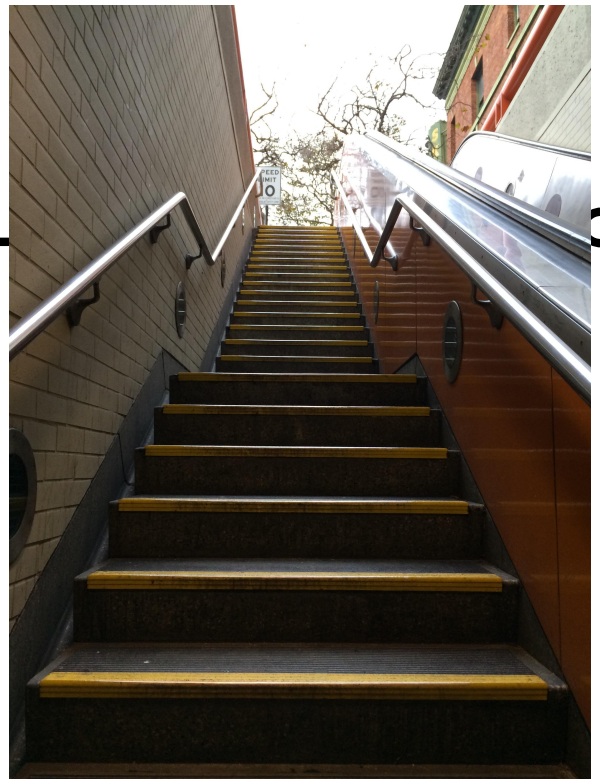
# A wonderful side effect to creating and using objects

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



# Objects

```
out.pri                                    orld!”);
```



# 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`

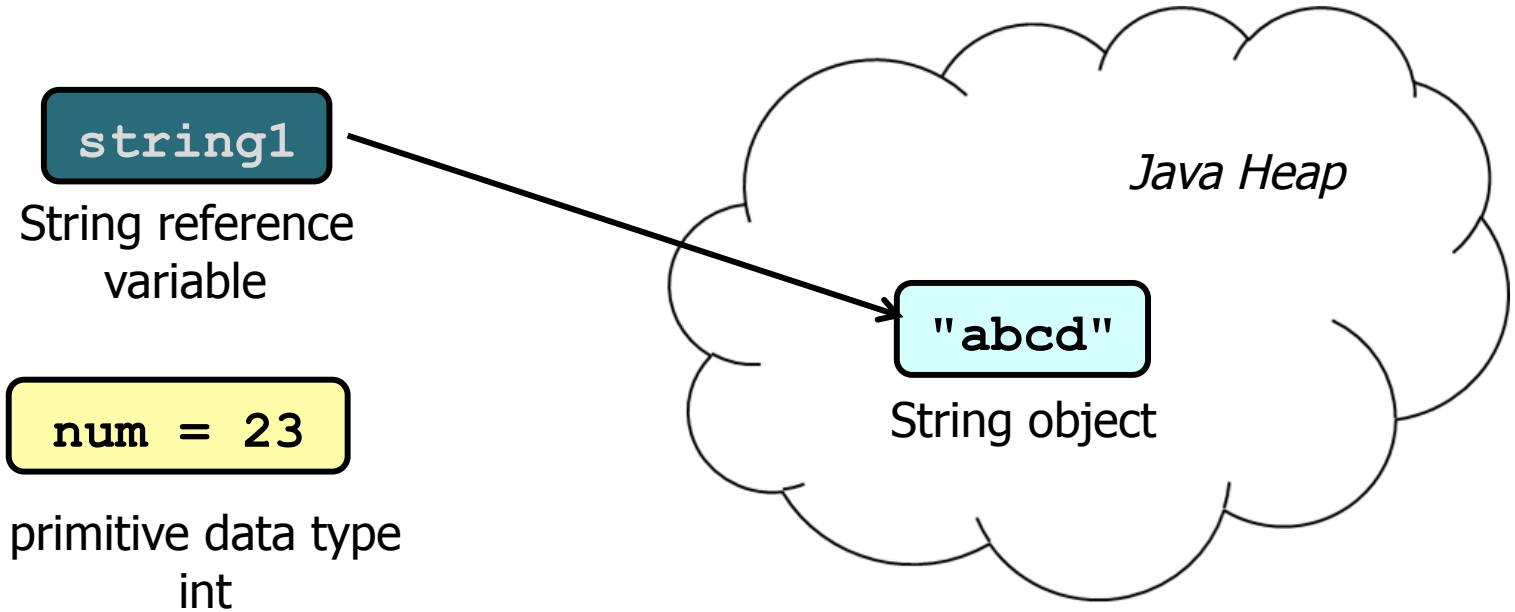
**String *object***

- "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

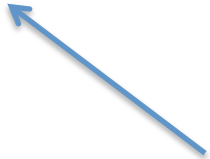
*Prints "wxyz" String literal*

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

## int primitive data type

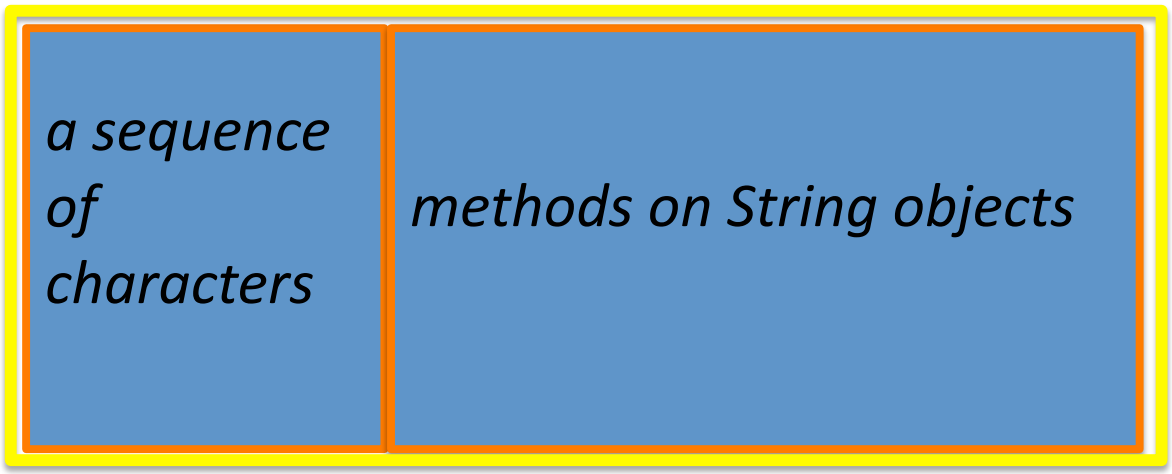
*Prints 27*

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



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

Here is where the similarities end



# Java String class

*Assign String literal "cde" to a variable*

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

*Concatenate & print "cdexyz"*

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

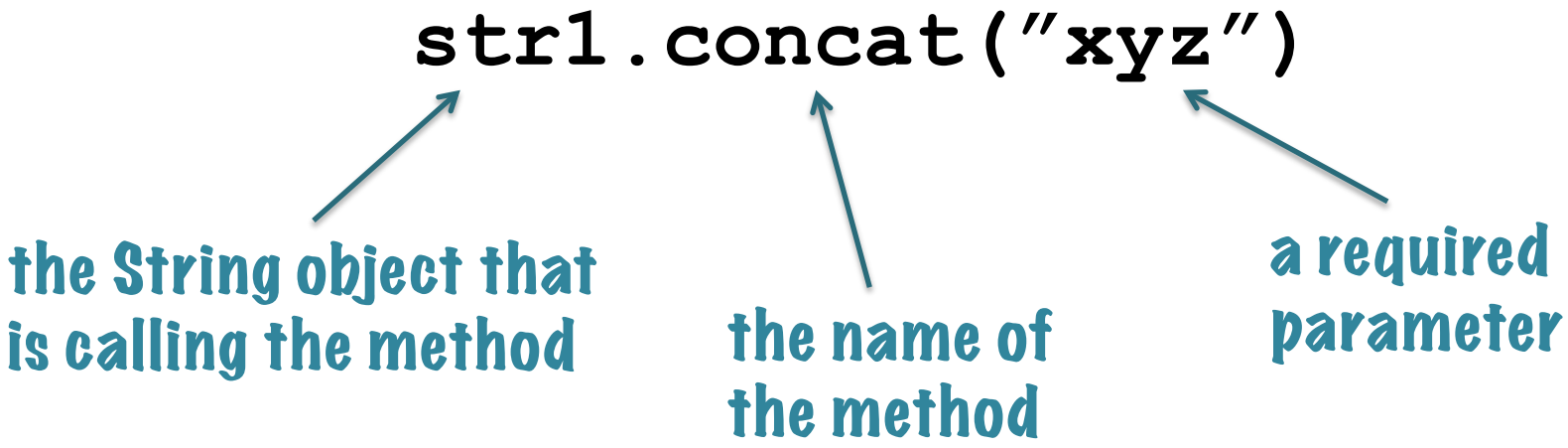
*Extract substring beginning at a and ending **before** c*

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

*Prints "ab"*

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

# Using methods with objects



# Using methods with objects

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

This method doesn't change str1 but 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");
}
```

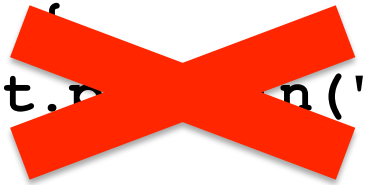
Never compare objects using ==
This will compile, but the results may not be what you are really asking for

# 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**