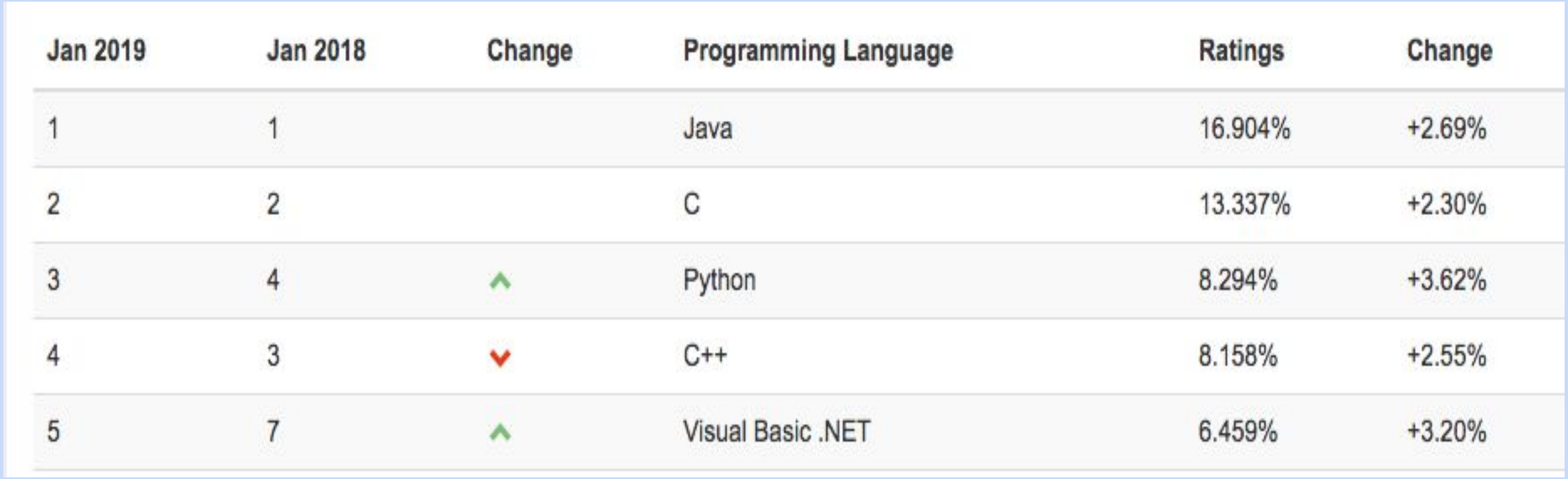
CS 342 Software Design

- Office hours start next week. Check office hours link for times.
- Teaching candidate on Tuesday at 12:30pm section of class.
- ACM Tech News: Java, Development, Design and PM

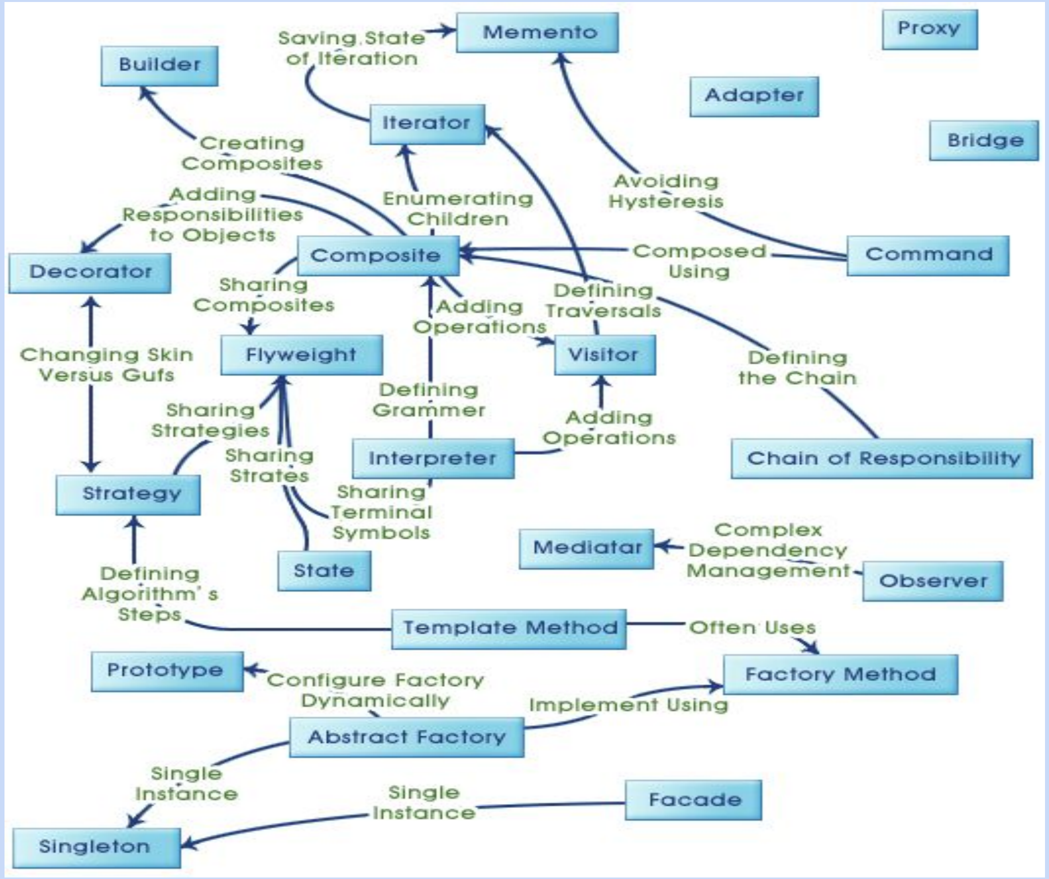
Java: Why are we learning it?

TIOBE Index for January 2019



Jan 2019	Jan 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.904%	+2.69%
2	2		C	13.337%	+2.30%
3	4	▲	Python	8.294%	+3.62%
4	3	▼	C++	8.158%	+2.55%
5	7	▲	Visual Basic .NET	6.459%	+3.20%

Java is full of design patterns!!!

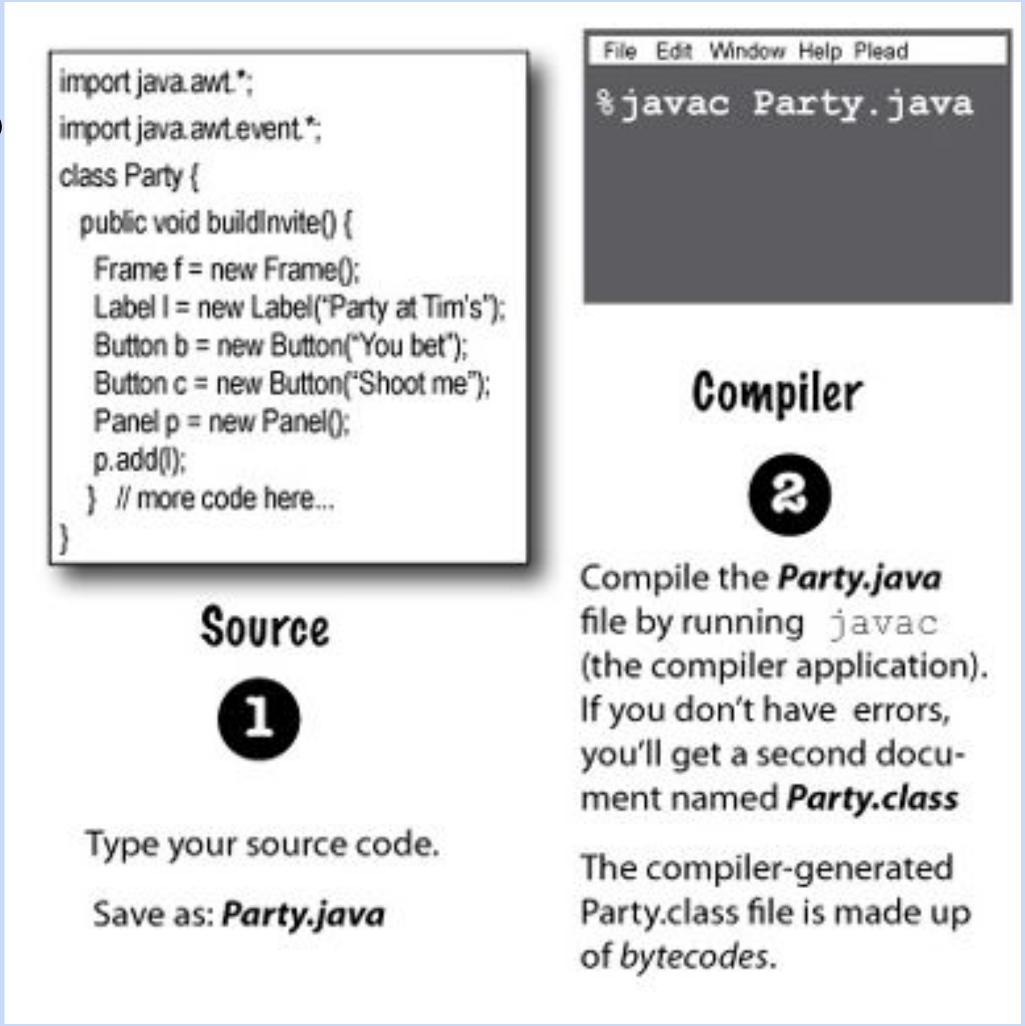


How does it work?

Bytecode is an intermediate, machine independent representation of a program.

Other Languages that compile to bytecode:

C#, F#, Kotlin, Python, .Net Framework…….



```
import java.awt.*;
import java.awt.event.*;
class Party {
  public void buildInvite() {
    Frame f = new Frame();
    Label l = new Label("Party at Tim's");
    Button b = new Button("You bet");
    Button c = new Button("Shoot me");
    Panel p = new Panel();
    p.add(l);
  }  // more code here...
}
```

```
%javac Party.java
```

Type your source code.

Save as: ***Party.java***

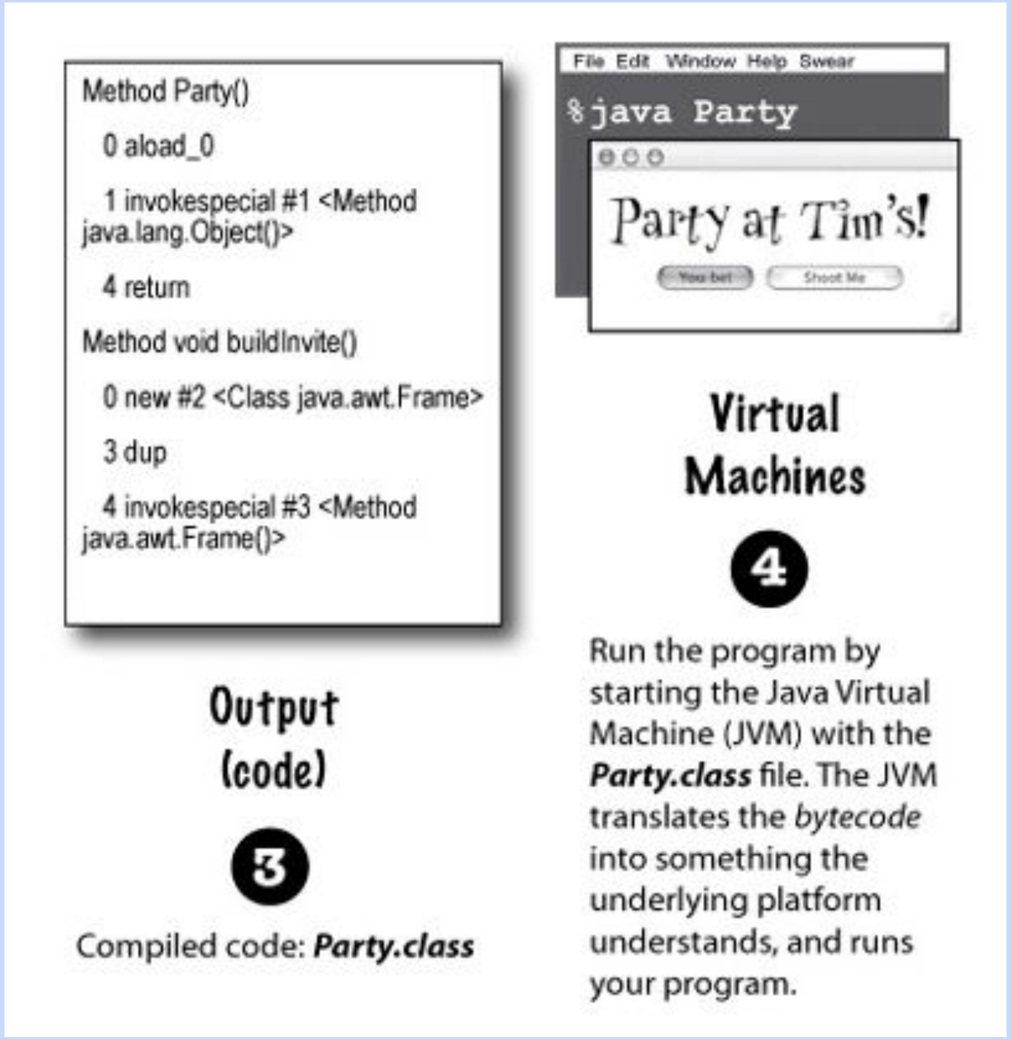
Compile the ***Party.java*** file by running `javac` (the compiler application). If you don't have errors, you'll get a second document named ***Party.class***

The compiler-generated Party.class file is made up of *bytecodes*.

How does it work?

The JVM handles the specifics of the platform the code is running on.

The programer doesn't need to think about what operating system or processor the code is running on.



```
Method Party()
  0 aload_0
  1 invokespecial #1 <Method
java.lang.Object()>
  4 return
Method void buildInvite()
  0 new #2 <Class java.awt.Frame>
  3 dup
  4 invokespecial #3 <Method
java.awt.Frame()>
```

```
%java Party
```

Compiled code: ***Party.class***

Run the program by starting the Java Virtual Machine (JVM) with the ***Party.class*** file. The JVM translates the *bytecode* into something the underlying platform understands, and runs your program.

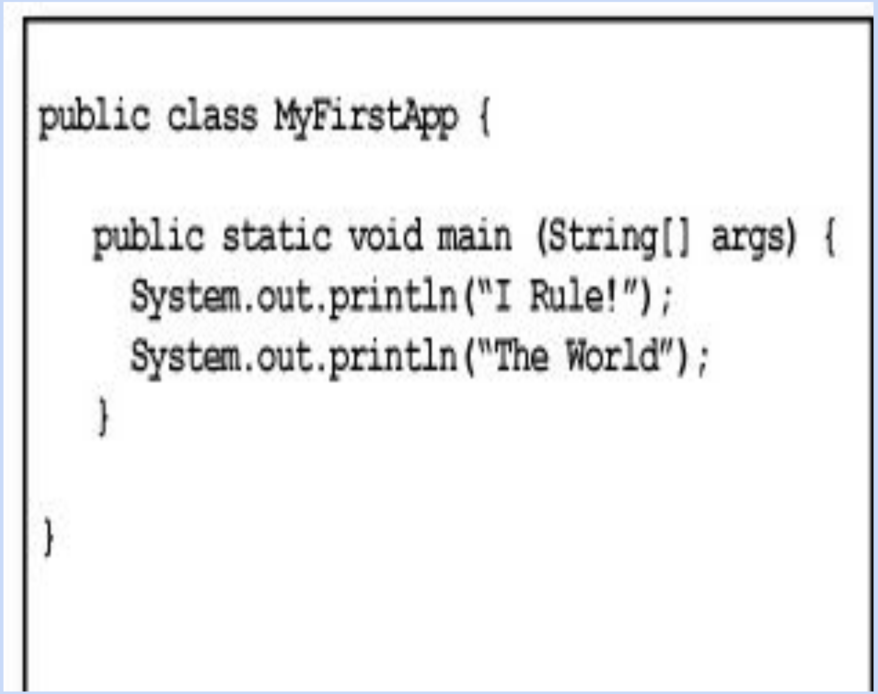
Java: Basic Program

(Save) MyFirstApp.java

(Compile) Javac MyFirstApp.java

    (produces) MyFirstApp.class

(Run) java MyFirstApp



```
public class MyFirstApp {

   public static void main (String[] args) {
     System.out.println("I Rule!");
     System.out.println("The World");
   }

}
```

The JVM will look for main in the class that you load.

CS 342 Software Design

- Clickers today!!!
- Classes
- Modifiers
- Methods

Java: Variables come in two types

Primitive types hold fundamental values: boolean and numeric.

Reference types hold classes, interfaces and arrays

Java is both ***Statically*** typed and ***Strongly*** typed!

Statically typed: every variable and expression has a type known at compile time

Strongly typed: limit value of variable and expression and limit operations supported

Java: Primitive variables

Integral types: byte, short, int, long, char

Floating point types: float, double

Boolean and char types: true, false, char

All primitives have a predefined size as well as defined operators and operations.

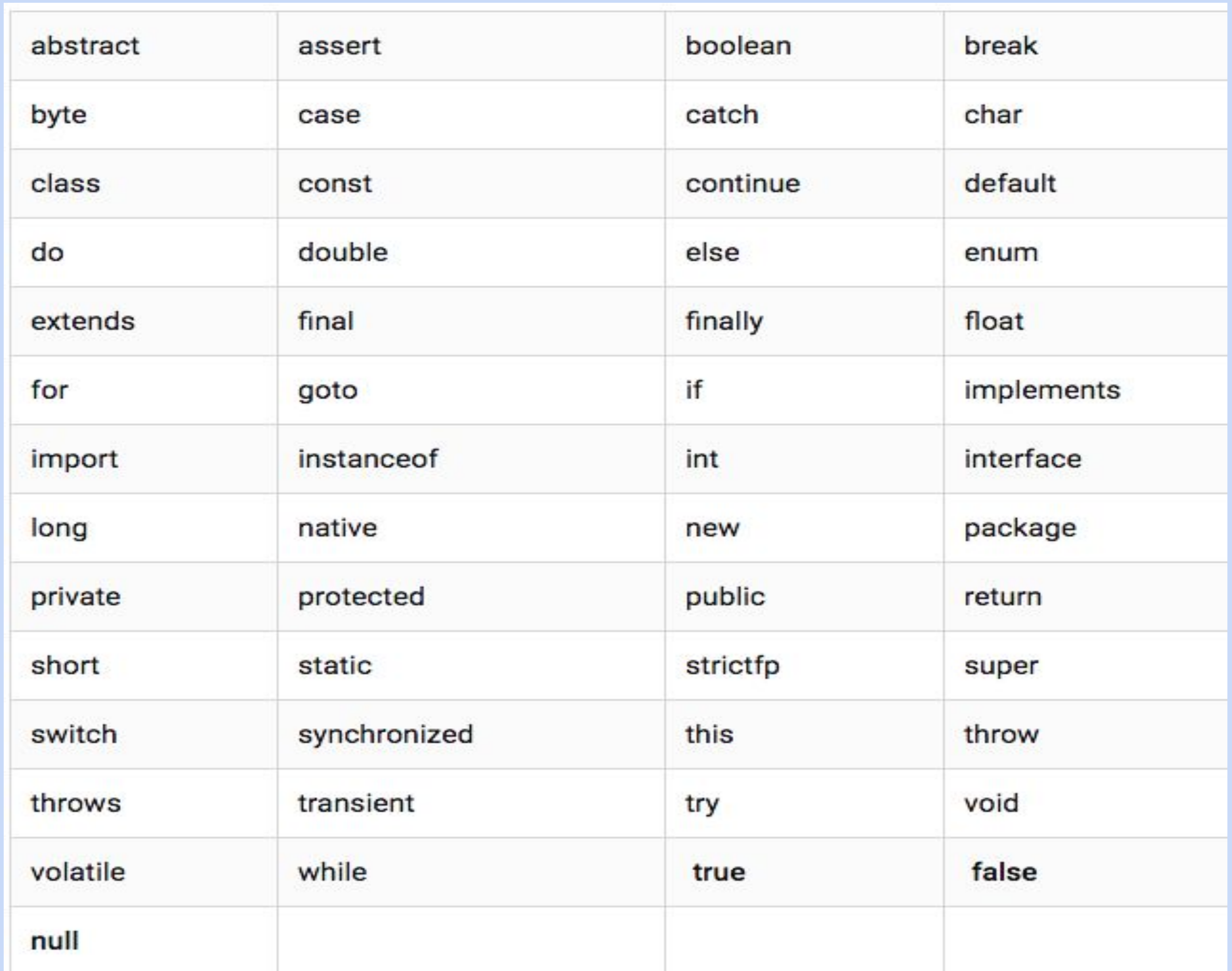
Java does not support operator overloading

Clicker Question: what is the result of this code?

```
public class q1{
    public static void main(String args [] ){
        int x = 24;
        byte y = x;
        System.out.println(y);
    }
}
```

A) 24
B) Runtime error
C) Won’t compile

Keywords:



abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while	**true**	**false**
null			

Java: Object references

These are variables that hold bits that represent a way to access an object.

It’s a way to get to an object.

```
Dog d = new Dog();

d.bark();
```

Use the object referenced by d to invoke the method bark

A Java reference is different than a C/C++ style pointer!

No pointer arithmetic: fundamental security feature in Java!

No re-casting! Object reference must actually be that type of object.

The Java Class:

- A class is a blueprint for an object. It tells the virtual machine how to make an object of that particular type.
- It contains instance variables, methods and more
- Can define a constructor or not
- The Java class has single inheritance but can implement multiple interfaces.

Simple example of a class!

```
class Dog{

	private int size; //instance variable

	private String name; //instance variable

	public Dog(int val, String val2){ //constructor

		this.size = val;

		this.name = val2;

	}

	public void bark(){ System.out.println(“woof...woof”);}

}
```

```
// in main

// two instances of class Dog
Dog d1 = new Dog(20, Rex);
Dog d2 = new Dog(21, Sasha);

d1.bark();
d2.bark();
```

Clicker: What are the two types of Java variables?

A) Object types and primitive types.
B) Reference types and object types.
C) Static types and strong types.
D) [✓ Reference types and primitive types.]

Clicker: What does strongly typed mean?

A) Every variable and expression has a type at compile time.

B) Variables can always exceed their stated capacity.

[✓] There is a limit to the value of each variable and expression.

D) There is no limit to the operations supported.

Classes: Encapsulation (fundamental OOP concept)

- Data and methods acting on it in a single unit
- All classes are subclasses of Object:
  - Documentation: https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html
- Two basic types: Top level, Nested
- Single inheritance model
- Access Modifiers

Classes:

- Body of a class declares: fields, methods, nested classes and interfaces, instance and static initializers and constructors.
- Class access modifiers: public, private, protected
- Other modifiers: static, abstract, final

Classes: modifiers

- **public**: Access from anywhere
- **private**: Access only from class where declared(nested)
- **abstract**: has methods that are declared but not implemented, can not be instantiated only extended by a subclass
- **final**: Can not have subclasses. Allows for optimization.
- **static**: only a nested class: more on nested classes to come!
- **default**: package access

Classes: Inheritance (fundamental OOP concept)

- Inherit the fields and methods of another class with the ability to add your own.
- Class members are inherited from direct superclass, direct super interfaces and body of the class if access modifier is public or protected.
- Not inherited are constructors, static and instance initializers and private members(needs getter/setter)

Clicker Question: What can I say about this code?

```
class Point { int x, y; }
class ColoredPoint extends Point { int color; }
final class Colored3dPoint extends ColoredPoint { int z; }
```

A) The class Point is a superclass of class Colored3dPoint.

B) The class ColoredPoint is a superclass of class Colored3dPoint and subclass of Point.

C) The class Colored3dPoint is a subclass of class ColoredPoint.

D) The class Colored3dPoint is a subclass of class Point.

[✓ E) All of the above]

Classes: constructors

- Block of code called when instance of an object is created
- Constructor modifiers: public, protected or private
- Can be overloaded: same name but different signatures
- Default constructor if none specified: for example

```
public class Point {
    int x, y;
}
```

```
public class Point {
    int x, y;
    public Point() {
super(); }
}
```

Q & A: is there anything wrong with this code? Click A when you have posted.

```
class Point { int x, y; }
final class ColoredPoint extends Point { int color; }
class Colored3DPoint extends ColoredPoint { int z; }
```

Final optimization

Original….new stack frame for each set() call in the loop

```
final class Point {
    int x, y;
    void set(int dx, int dy) { x += dx; y += dy; }
}
class FinalTest {
    public static void main(String[] args) {
        Point[] p = new Point[100];
        for (int i = 0; i < p.length; i++) {
            p[i] = new Point();
            p[i].set(i, p.length-1-i);
        }
    }
}
```

Final optimization

Optimized!….no method calls, reduced overhead.

```
for (int i = 0; i < p.length; i++) {
    p[i] = new Point();
    int j = p.length-1-i;
    p[i].x += i;
    pi.[y] += j;
  }
```