All programming constructs in Java, including methods, are part of a class

A constructor is called when an object of the class is created using   
new ClassName objectName = new ClassName(anyArgs);  
  
Java will automatically create a default or no-argument constructor This allows the object to be created

Overloading is when two or more methods in the same class have the same method name •  
  
 To be valid, any two definitions of the method name must have different signatures A signature consists of the name of a method together with its parameter list Differing signatures must have different numbers and/or types of parameters

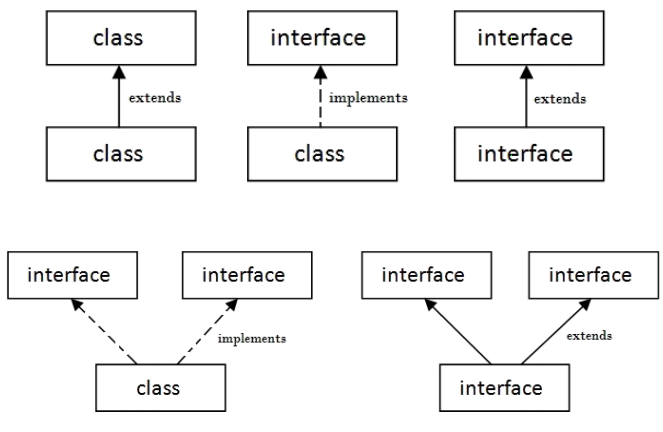
**Public class** should provide everything users need, but nothing they don’t. Access Modifiers- Need2know principle

**Accessor** methods allow the programmer to obtain the value of an object's instance variables   
The data can be accessed but not changed The name of an accessor method typically starts with the word get  
  
**Mutator** methods allow the programmer to change the value of an object's instance variables   
Incoming data is typically tested and/or filtered The name of a mutator method typically starts with the word set  
  
**Instance variables** are declared in a class, but outside a method, constructor or any block. When a space is allocated for an object in the heap, a slot for each **instance variable** value is created have . in front of them   
  
If an explicit name for the calling object is needed, the keyword this can be used myInstanceVariable always means and is always interchangeable with this.myInstanceVariable  
  
**This** parameter is a kind of hidden parameter   
  
**Final** - keyword is similar to Const in C++ Locks the data.  
Primitive values stay constant. References always refer to same object.  
  
**Class Reuse-** Composition is called “has-a”. Inheritance is called “is-a”.  
  
**Static variable** is a variable that belongs to the class as a whole, and not just to one object **Class Variable**! There is only one copy of a static variable per class, All objects of the class can read and change a static variable

**All parameters** in Java are call-by-value parameters A parameter is a local variable that is set equal to the value of its argument Therefore, any change to the value of the parameter cannot change the value of its argument

**Class type** parameters appear to behave differently from primitive type parameters They appear to behave in a way similar to parameters in languages that have the call-by-reference parameter passing mechanism

The **== operator** only checks that two objects have the same memory address   
  
**Garbage collection** is automatic, manual call with System.dc();   
  
**String Class** is array of chars.   
  
Never write a method that returns a mutable object Use copy constructor to return refernce.  
  
  
**Information hiding/Abstraction** is the practice of separating how to use a class from the details of its implementation  
**Encapsulation** means that the data and methods of a class are combined into a single unit (i.e., a class object), which hides the implementation details. Done by making them private.  
**Inheritance** is the process by which a new class is created from another class The new class is called a derived class The original class is called the base class   
  
**Sub/Child** inherits from the super/base class  
  
**Covariant Return Type** -the returned type may be changed to that of any descendent class of the returned type   
  
**Override**, the new method definition given in the derived class has the exact same number and types of parameters as in the base class. Argument and return type must match. Late binding performed at runtime.   
  
**Overload**• When a method in a derived class has a different signature from the method in the **Base class**, still inherits the original method from the base class   
  
**Final** - method may not be be redefined in derived class. Class cannot be used as base either.

**Super** - Call a superclass method from the subclass. Compiler calls this if not specified. Requires no arg constructor. Cannot super.super.something()  
  
**This** can be used as a name for invoking another constructor in the same class. Points to instance variable rather than the local.   
  
**Private Methods -**  Cannot be inherited, Can only be called directly.  
  
**Protected** - Inside its own class definition Inside any class derived from it. In the definition of any class in the same package. Weaker than private  
  
**Package Access Modifier** - Variable/Method without a preceeding modifier. Can be accessed anywhere inside package.  
  
**Static** - Belongs to the class and not the object it is in.  
  
**Class Object** - every class is a descendent of the class Object. Every object of every class is of type Object, as well as being of the type of its own class  
  
**Typecast** - Assigning different variable type.   
  
**Abstract Method** - placeholder. Can’t be private, complete heading, with no body. Abstract any number of methods. can have abstracts derived classes.Abs Class Const can’t be used to create an obj of abs class. Super is passed as abs  
  
**Concrete Class** - no abstract methods.  
  
**Interface** - Extreme abstract class. Designed to specify methods the concrete class will use. It is not a class. Interface can have no data members. Multiple Inheritance from base classes not allowed. Can only be public.  
  
**Implements** -a concrete class must include Implements at start of class def. Must implement all method headings.  
  
**Extends** -may be derived from a base interface using Ex tends. Concrete class that implements a derived interface must have definitions for any methods in the derived interface as well as any methods in the base interface   
  
  
  
**Comparable -** Checks if same type.  
  
  
**Object serialization** - sequence of bytes that includes the object's data, object's type and the types of data. No headings, no constants. Empty.Indicates file I/O   
  
 **Object deserialization**: reads the object data, the type information and bytes that represent the object from the file it was written in and to recreate the object in memory.  
  
**Inner Class** - Can reference private variable/method in outer.Outer can reference inner private variable, or class if inner class is calling object.  
  
**ClassFiles-** Compiling any class in Java produces a .class file named ClassName.class  
  
**Thread** - Seperate computation process, cpu randomly switches resources.Start calls run.  
  
**Sleep** - Pauses the thread for milliseconds. May throw InterruptedException  
  
**Runnable** - Also makes a thread. When you extend Thread class, each of your thread creates unique object and associate with it. When you implement Runnable, it shares the same object to multiple threads.  
  
**Race Condition** - multiple threads change a shared variable.  
  
**Synchronized** - only one thread can be run at a time to prevent RC.  
  
**TCP** - Server waits for connection. Client initiates. Connects over Socket 0-65535. One program per port. 0-1024 are reserved.  
  
**accept()** call is typically placed in a loop and a new thread created to handle each client connection:  
  
**Polymorphism** is the ability to associate many meanings to one method name It does this through a special mechanism known as late binding or dynamic binding  
  
**Late binding** for all methods (except private, final, and static methods)   
  
**Upcasting** is when an object of a derived class is assigned to a variable of a base class (or any ancestor class)  
  
**Downcasting** is when a type cast is performed from a base class to a derived class (or from any ancestor class to any descendent class). Careful, danger!  
  
**Collections** - Holds objects and implements the collection interface.ArrayList<T> is example. Implements all methods of Collections. Used with iterators.  
   
**Wildcards -** collection framework can have parameter type specifications that do not fully specify the type plugged in for the type parameter. Specify wide range of argument types  
  
**Collection<t>** framework can have parameter type specifications that do not fully specify the type plugged in for the type parameter  
  
**List<T>**: Provides ordered and indexed collection which may contain duplicates. Ordered, Duplicates, Positional, Null Duplicates. =Arraylist, LinkedList, Vector  
**Set<T>**: A collection that contains no duplicate elements. Unordered, No Duplicates, NoPosition, 1 Null, Hashset, LinkedHashset, Treeset.  
  
  
**ArrayList-** AL is newer and faster.ArrayList is implemented using an array as a private instance variable. When full moves to new array, autogrow but not shrink. Only Class. Not primitive.  
  
ArrayList list.add("something"); //Add is overloaded.  
int howMany = list.size(); //find size   
list.set(index, "something else");   
String thing = list.get(index);  
  
**Iterating Collections-** check hasNext() boolean, before using next(). Don’t modify while iterate.