***@ Understanding Class with example***

Think you self as the god. And you can do everything. Now you want to create an object in this new world. For example, human beings. However, before you create human, you have to design them first. And this design blueprint is the **class** in Java. Every human in this world have to follow this blueprint, or, has the feature written on this blueprint.

In this blueprint, you define a lot of detailed things. For example, what do human have and what can they do? Human should have a name, a sex type (male or female), 1 head, 2 legs and 2 arms, etc. And they can walk, they can run, and some of them can even swim. They thing they have are what's called **attributes** or data fields. And what they can do are known as **behavior** or methods in Java.

so you may define a class named Human in Java like this:

1. public class Human {
2. String name;
3. int HP = 100;
4. int stamina = 100;
5. String sex;
6. public Human(String name, String sex) { // the constructor
7. this.name = name;
8. this.sex = sex;
9. }
10. public void run(){ // a method called run
11. stamina -= 10;
12. }
13. public void sleep(){ // a method called sleep
14. stamina += 10;
15. }
16. }

Take this class for instance. If you use this class as the blueprint of Human, then every human would have 100 in HP, 100 in stamina when you create them. And besides, every human can run and sleep. Run will cost the stamina and sleep can recover the energy. However, this is just a blueprint. To create a human, you have to say something like:

Human adam;

That's how you create a new object of Human. Notice here we did not use the constructor. Why do you -- the god, need the constructor? Since you only defined the default value of HP and stamina but name and sex, you have to use constructor to set these two value for every object using parameters:

Human eva = new Human("Eva", "Female");

And this Human "Instance(or Object)" eva, got a name Eva, and she is a female. She can run and sleep, as you defined in the Human class.

notice here "eva" isn't the name of this object, but a reference, a nickname. When you need to use this object, you have to use this nickname to cite it. But the actual name you set for this human is what you said in the constructor -- "Eva".

If you still have confusion, check The Java Tutorials:  
http://docs.oracle.com/javase/tutorial/java/concepts/class.html

***# Use of ‘this’ keyword***

this is a reference to the current object.

Imagine you have a class named Person.

1. class Person{
2. String name;
3. public void setName(String name){
4. this.name = name;
5. }
6. }

There are 2 variables with the name 'name', one is an attribute String name; of the class, so all objects of the class will have a name, and the other is an argument for the method setName(String name).

this.name = name; inside of the method, means that we are giving the value of the name passed as argument to the attribute name, so the value passed will be the name of the object. Is as if this is taking you out of all methods and make you access the members of the object, in this case String name;

If we create two objects:

1. Person p1 = new Person();
2. Person p2 = new Person();

Then assign names to them with the method:

p1.setName("Ben");//'this' inside this call will point to object p1

p2.setName("Liam");//'this' inside this call will point to object p2

So 'this' means 'current object', and there names will be set:

p1 will have name Ben

p2 will have name Liam

***# Purpose of setting member variables to ‘Private’***

The keyword here is [Encapsulation](http://en.wikipedia.org/wiki/Encapsulation_%28object-oriented_programming%29). In OOP you want to use private variables in order to enforce proper encapsulation of your objects/classes.

While other programmers are not out to get you, they do interact with your code. If you do not make a variable private, they may well refer to it in their own code without wanting to do any harm at all. However, if you need to get back to your class and change something, you no longer know who uses which variable and where. The goal of encapsulation is to make the external interface of the class explicit so that you know only these (typically) methods could have been used by others.

1. Hence, private variables ensure that the corresponding variable remains in the defining class only. If you need to change it, the change is local to the class itself.
2. In traditional languages like C++ or Java, you usually make everything private and only accessible by corresponding getters and setters. No need to decide much really.
3. Sometimes, for example in a C++ struct, you need a class only as a way to group several things together. For example, consider a Vector class which has an x and a y attribute only. In those cases, you may allow direct access to these attributes by declaring them public. In particular, you must not care if some code outside modifies an object of your class by directly writing new values into x or y.

In those languages, you therefore get something called uniform access principle, where one deliberatly does not distinguish methods like getters and setters, but provides direct access to the variable/function. So you may say that private declarations are quite a lot more popular in the object-oriented scenarios.

// details explanation

If you work in team then your team members will use code that you wrote. I would not say that variables inside the class are local to that class, they are more like members of that class. There are many reasons to hide code from being public.

Private variables are good to avoid miss-usages leading the object into an inconsistent status and hard to track bugs and unforeseen exceptions.

Here is one example, lets say you have classes Company, Office and Employee which are defined as

1. public class Company {
2. public String name;
3. public List<Office> offices = new ArrayList<>();
4. }
6. public class Office {
7. public String name;
8. public Company company;
9. public List<Employee> employees = new ArrayList<>();
10. }
12. public class Employee {
13. public String firstName;
14. public String lastName;
15. public String email;
16. public String notes;
17. public Office office;
18. }

In Main class you have something like the following

1. public static void main(String[] args) {
2. Company company = new Company();
3. company.name = "MyCompany";
5. Office office = new Office();
6. office.name = "MyOffice";
8. company.offices.add(office);
10. // company has 1 office which is ok
11. // now the other team member comes and writes code like this
12. company.offices = new ArrayList<>();
14. // that will delete all existing offices in company and when you have 10k classes in project it is not easy
15. // to find where that bug is
17. // now the other team member comes and writes seomething like this
18. company.offices = null;
19. // again deleting all offices, setting office list to null, all methods in company class if there are any
20. // that depend on that list will cause null pointer exception and code is more or less broken
22. // now again you come back to your code and you want to add another offices
23. company.offices.add(new Office);
24. // that will cause null pointer exception because somebody before you set offices to null
25. // now again you have another bug in the code that was originally cause by you but others team mates
26. // wrote tons of code that depends on your code, meaning that all your code base is broken
27. }

The same goes for the list of employees in Office.

On the other hand you classes need some more functionality, lets say you want to know full name of employee, somebody will say just create another String field in Employee class and call it fullName, that makes another variable that you have to set, if using databases you will need another column in database, your database becomes bigger and bigger because of that variable.

So your employee class becomes this

1. public class Employee {
2. public String firstName;
3. public String lastName;
4. public String email;
5. public String notes;
6. public Office office;
7. public String fullName
8. }

Which does not make any sense, why would you add fullName when fullName is firstName + lastName you can just create a method that will create string for you. The requirement is that fullName needs to be firstName + " " + lastName and if last name is null or empty you dont need the space between them. That functionality obviously belongs to Employee class. Two weeks after comes another teammate and he needs to know in which company is employee working, and he would say lets add Company to Employee and employee class becomes the following

1. public class Employee {
2. public String firstName;
3. public String lastName;
4. public String email;
5. public String notes;
6. public Office office;
7. public String fullName
8. public String company
9. }

Now again that does not make much sense, since through the office you can get company to which office belongs. This will again take additional variable, again another column in database, more memory etc. As you can see company has many offices, each offices can have many employees (notice there is a List). Each Employee can belong to only 1 office and each office can belong to only 1 company. So to link office with company you would need to write something like this

1. Company company = new Company();
2. company.name = "myCompany";
3. Office office = new Office();
4. office.name = "myOffice";
6. // link office and company
7. company.offices.add(office);
8. office.company = company;
10. // now you want to remove it
11. company.offices.remove(office);
12. office.company = null;

That is a lot of code and it could easily generate mistake if somebody accidentally deletes all offices or sets list to null. How many times you will have to add new office lets say around 100, what about employees some companies could have 10k employees repeating code to add employee to office will not lead you anywhere.

Now lets switch to private members.

1. public class Company {
3. private String name;
4. private final List<Office> offices = new ArrayList<>();
6. public void setName(String name) {
7. this.name = name;
8. }
10. public String getName() {
11. return name;
12. }
14. public void addOffice(Office office) {
15. // automatically adds and links company with office
16. // if you just add it to list company will be null in office and that is not what you want
17. office.setCompany(this);
18. offices.add(office);
19. }
21. public void removeOffice(Office office) {
22. // automatically removes and unlinks company with office
23. // if you just remove it from list there is corrupted data
24. // since office still has company but it office is not in list of offices
25. office.setCompany(null);
26. offices.remove(office);
27. }

30. public class Office {
31. private String name;
32. private Company company;
33. private final List<Employee> employees = new ArrayList<>();
35. public void setName(String name) {
36. this.name = name;
37. }
39. public String getName() {
40. return name;
41. }
43. public Company getCompany() {
44. return company;
45. }
47. public void addEmployee(Employee employee) {
48. // automatically adds and links employee with office
49. employee.setOffice(this);
50. employees.add(employee);
51. }
53. public void removeEmployee(Employee employee) {
54. // automatically removes and unlinks employee from office
55. employee.setOffice(null);
56. employees.remove(employee);
57. }
59. // useful method to have since list is not exposed to public
60. public Integer getEmployeeCount() {
61. return employees.size();
62. }
64. public class Employee {
65. private String firstName;
66. private String lastName;
67. private String email;
68. private String notes;
69. private Office office;
70. private boolean active = true;
72. // getters/setters ommited
74. // method to find at which company is this employee working
75. public Company getCompany() {
76. return office == null ? null : office.getCompany();
77. }
79. public String getFullName() {
80. // this needs better implementation but for now just assume they are not null
81. return firstName + " " + lastName;
82. }

You should see the difference, now nobody can delete employees or offices by accident, since list is hidden from public. But classes provide useful methods to add employee/office and they automatically link one to each other so there is no need to repeat the same code when adding new employees/offices. You can see how exceptions and errors are avoided when you have private members.

Imagine having sorting alghoritm class that sorts elements and somebody changes public member all sorting is automatically broken.

Hopefully this should clear the things up. Sorry for the long answer.

Cheers.

*Q) are objects and Instances the same thing?*

Ans>

Creating an object means creating a specific member or instanciate a specific object of that type of objects. An object-class is the blueprint how to create an object. The created object is the instance of that object. All instances of a object class differs of the values of their instance variables. I do not like the word instance, I prefer member of that class.

*Q) Difference Between Class / Object / Instance / Reference*

Ans>

**Object**  
Real world objects shares 2 main characteristics, *state* and *behavior*. Human have state (name, age) and behavior (running, sleeping). Car have state (current speed, current gear) and state (applying brake, changing gear). Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in *fields* and exposes its behavior through *methods*.

**Class**  
Class is a “template” / “*blueprint*” that is used to create objects. Basically, a class will consists of field, static field, method, static method and constructor. Field is used to hold the state of the class (eg: name of Student object). Method is used to represent the behavior of the class (eg: how a Student object going to stand-up). Constructor is used to create a new Instance of the Class.

**Instance**  
An instance is a unique copy of a Class that representing an Object. When a new instance of a class is created, the JVM will allocate a room of memory for that class instance.

*Q) Can we use class as a data type*

Ans>

Data type isn't restricted to just primitive data type like int, double, long, etc.  It can be String or Object.  Remember, a Class is like an object constructor, or a "blueprint" for creating objects with fields and methods.  So that's the power of OOP is that you can **have the Objects/Classes interacting with each other** by using Object or Class as your data type.  So in exercise 34 of my sample code below:

1. public class ComplexNumber {
2. public void add(ComplexNumber complexNumber){
3. real=real+complexNumber.getReal(); // go to Class/Object ComplexNumber below and call the method .getReal()
4. imaginary=imaginary+complexNumber.getImaginary(); // go to Class/Object ComplexNumber below and call the method .getImaginary()
5. }
6. }

I'm separating these code within the Class just for demonstration...

1. public class ComplexNumber {
2. private double real;
3. private double imaginary;
5. public double getReal(){
6. return real;
7. }
8. public double getImaginary(){
9. return imaginary;
10. }
11. }

This is saying that for the method add(), I'm going to pass one parameter of data type Object, specifically ComplexNumber with uppercase C because it's a Class, and I'm going to name the parameter complexNumber with lowercase c.  Then in the code block, I'm going to refer to that Object/Class by using the reference variable complexNumber to get the method getReal() by using complexNumber.getReal().  Likewise, we can call the getImaginary() method in that same Object/Class by using the reference variable  complexNumber and the method with complexNumber.getImaginary().

Perhaps the confusing part was that the Object/Class data type referenced was the Object/Class itself.  Had this Object been in a different .java file,  the data type would have instructed the program to look in that "Object/Class" or .java file and get the method there.  This is why OOP is so powerful is that you are able to define the interactions between Objects through referencing to the Class, whether it is in that same or different .java file.  Hope that makes sense.

Here is my complete code/solution for exercise 34 with everything in just one Class of file in ComplexNumber.java.

1. public class ComplexNumber {
2. private double real;
3. private double imaginary;
5. public ComplexNumber(double real, double imaginary){
6. this.real=real;
7. this.imaginary=imaginary;
8. }
9. public double getReal(){
10. return real;
11. }
12. public double getImaginary(){
13. return imaginary;
14. }
15. public void add(double real, double imaginary){
16. this.real=this.real+real;
17. this.imaginary=this.imaginary+imaginary;
18. }
19. public void add(ComplexNumber complexNumber){
20. real=real+complexNumber.getReal();
21. imaginary=imaginary+complexNumber.getImaginary();
22. }
23. public void subtract(double real, double imaginary){
24. this.real=this.real-real;
25. this.imaginary=this.imaginary-imaginary;
26. }
27. public void subtract(ComplexNumber complexNumber){
28. real=real-complexNumber.getReal();
29. imaginary=imaginary-complexNumber.getImaginary();
30. }
31. }