# What is polymorphism ?

# Polymorphism means one interface and many forms. Polymorphism is a characteristics of being able to assign a different meaning or usage to something in different contexts specifically to allow an entity such as a variable, a function or an object to have more than one form. There are two types of Polymorphism. Compile time: function or operator overloading Runtime: Inheritence & virtual functions

# [What is the difference between abstract function and virtual function?](http://stackoverflow.com/questions/391483/what-is-the-difference-between-abstract-function-and-virtual-function)

An abstract function can have no functionality. You're basically saying, any child class MUST give their own version of this method, however it's too general to even try to implement in the parent class. A virtual function, is basically saying look, here's the functionality that may or may not be good enough for the child class. So if it is good enough, use this method, if not, then override me, and provide your own functionality.

An abstract function has no implemention and it can only be declared on an abstract class. This forces the derived class to provide an implementation. A virtual function provides a default implementation and it can exist on either an abstract class or a non-abstract class. So for example:

public abstract class myBase  
{  
    //If you derive from this class you must implement this method. notice we have no method body here either  
    public abstract void YouMustImplement();  
  
    //If you derive from this class you can change the behavior but are not required to  
    public virtual void YouCanOverride()  
    {   
    }  
}

**What is an Interface ? What is an abstract class ?**

An interface looks like a skeleton or an outline of the class.

An interface is a contract & defines the requisite behavior of generalization of types.   
An interface mandates a set of behavior, but not the implementation. Interface must be inherited. We can't create an instance of an interface. An interface is an array of related function that must be implemented in derived type. Members of an interface are implicitly public & abstract.   
An interface can inherit from another interface.

Abstract class is a class that can not be instantiated, it exists extensively for inheritance and it must be inherited. There are scenarios in which it is useful to define classes that is not intended to instantiate; because such classes normally are used as base-classes in inheritance hierarchies, we call such classes abstract classes.   
  
Abstract classes cannot be used to instantiate objects; because abstract classes are incomplete, it may contain only definition of the properties or methods and derived classes that inherit this implements it's properties or methods.   
  
Static, Value Types & interface doesn't support abstract modifiers. Static members cannot be abstract. Classes with abstract member must also be abstract.

**The key differences between an** [**abstract class**](http://us2.php.net/abstract) **and an** [**interface**](http://us2.php.net/interface) **are:**

* Abstract classes can have consts, members, method stubs and defined methods, whereas interfaces can only have consts and methods stubs.
* Methods and members of an abstract class can be defined with any visibility, whereas all methods of an interface must be defined as public.
* When inheriting an abstract class, the child class must define the abstract methods, whereas an interface can extend another interface and methods don't have to be defined.
* A child class can only extend a single abstract (or any other) class, whereas an interface can extend or a class can implement multiple other interfaces.
* A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility.

**When to use Interface and when to use abstract class ?**

1. If base class needs to change again and again, make base class abstract. If you use interface code will break. If you plan on updating this base class throughout the life of your program, it is best to allow that base class to be an abstract class. Why? Because you can make a change to it and all of the inheriting classes will now have this new functionality. If the base class will be changing often and an interface was used instead of an abstract class, we are going to run into problems. Once an interface is changed, any class that implements that will be broken
2. Another general rule is if you are creating something that provides common functionality to unrelated classes, use an interface. If you are creating something for objects that are closely related in a hierarchy, use an abstract class.
3. Real time example when to use interface: IAnimal is probably not a good example for an interface, but IWalkable might be a good interface to have if your system has many things that can walk. You might have classes derived from Animal such as Dog, Cow, Fish, Snake. The first two would probably implement IWalkable, the latter two don't walk, so they wouldn't. Now you ask "why not just have another superclass, WalkingAnimal, that Dog and Cow derive from?". The answer is when you have something completely outside the inheritance tree that also can walk, such as a robot. Robot would implement IWalkable, but probably wouldn't derive from Animal. If you want a list of things that can walk, you type it as IWalkable and you can put all walking animals plus robots in the list. Now replace IWalkable with something more software-y like IPersistable, and the analogy becomes much closer to what you'd see in a real program.



If a number is power of 2

bool isPow2(int n){  
n=n<0?-n:n;  
return !(n & (n-1));  
}

binary addition

Add\_Num( int a, int b)  
{

If(b== 0)  
{  
Return;  
}

Int sum = a ^ b ;  
Int carry = ( a & b ) << 1;

Add\_num( sum, carry)

}

Given two unsigned integers, write an efficient function which returns the no. of bits needs to be flipped of one to generate the other.

int bit\_swaps\_required( int a, int b ) {  
unsigned int count = 0;  
for( int c = a ^ b; c != 0; c = c & c-1 ) {  
++count;  
}  
return count;  
}

BIT MANIPULATION

**IF infinite streams of 1's and 0's are coming to find the new number due to the next coming bit can be found as mentioned below: let n bits be read so far and value corresponding to that be no..  
now we get another bit say newbit...  
new value of no will be no\*2+newbit**

VERY VERY IMP CONCEPT

String s1 = "9";

String s2 = "3";

int a = s1.charAt(0) - '0' + s2.charAt(0) - '0' ;

System.*out*.println( a);

O/P = 12

Expedia phone

1. Coupling cohesion
2. Alternative to inheritance
3. Interface & abstract class
4. Spell checker
5. Customer, many cards. Retrieve all cards count
6. Synchronization
7. Merge sort & quick sort
8. Sort based on last name for an object Person(Fname, Lname)
9. Abstract class constructor
10. Hibernate Advantages
11. Design pattern
12. difference between C# and java

**dynamic programming, greedy algorithms, solving linear recurrences in logarithmic time, recursion, tree traversals, etc.**

HP phone

Pattern

Static class and method when to use

WCF abc

Indexing

Out and ref for function

Static class or Singleton?

I only use static classes if I'm convinced the details won't change over time -- a utility class for parsing strings is a good example. If the code contains business logic, which is likely to change over time, I prefer the flexibility of the singleton approach.

<http://www.dotnetperls.com/singleton-static>

**SDE Guru:** What would you consider as your most challenging project?  
**Candidate NR:** At my previous employer, we had a large comples system. It was very bloated and over-engineered. It was implemented in the J2EE stack , with multiple layers of inderction. The entire application was poorly designed, while it was functional, it suffered from very major performance issues. Since, I had joined only recently, it was a delicate situation. After much thought, I decided to voice my concerns, and was pleasently surprised to hear that there were many in my team with similar opinion. I took a simple incremental approach to fixing the problem

**SDE Guru:** Good! You mentioned Java as your primary language, How do you keep yourself up-to-date?  
**Candidate JB:** Honestly, it is really hard to keep up to date. As a part of my regular work, we do not get to explore or work on cutting edge technology. In order to keep up to date, I regularly followup on a few blogs like .... and ... . In addition to this, I follow the progress in key conferences like the Java ONE. Surprisingly, I follow an open source group called .... that does great work on implementing simple alternative data mining methods. They happen to use Java as the language, and turns out that the smart developers behind the project tend to keep up with happenings in the Java World.

**Programming Interview 1: Reverse a Linked List**

Amazon onsite

* 1. design system for Amazon fresh/market
  2. Implement Hashtable- retrieve the list as values
  3. New data structure using hashtable + Array

Ebay Test Telephone

* + 1. Composition-aggregation, is-a and has –a
    2. Sort string char array
    3. Readfile, chance accnum with e to pp – write to a new file

# [What is the difference between aggregation, composition and dependency?](http://stackoverflow.com/questions/1644273/what-is-the-difference-between-aggregation-composition-and-dependency)

**Aggregation** implies a relationship where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.

**Composition** implies a relationship where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

The above two are forms of **containment** (hence the parent-child relationships).

**Dependency** is a weaker form of relationship and in code terms indicates that a class uses another by parameter or return type.

Dependency is a form of **association**.

1. A House **is a** Building (inheritance);
2. A House **has a** Room (composition);
3. A House **has an** occupant (aggregation).

The difference between (2) and (3) is subtle yet important to differentiate. Together they are forms of*association*. What's the difference? Composition implies the child object cannot live out of the context of the parent (destroy the house and rooms disappear) whereas aggregation implies the child can exist on its own (destroy the house and the occupant goes elsewhere).

**LinkedIn Phone Interview…**

Find the Deepest Common Ancestor of the Binary Tree where only two nodes are given and link to their parent is given but the root node is not given

**IMP funda for Min & Max Heap**

To **find top 3** numbers out of an array for example - [1,2,100,7,5,77,9]

- **Create a Min Heap of 3**

- Add numbers one by one and compare with the Top Element

- Top element in Min Heap is the smallest element in the heap

- **If the num you compared with the Top is greater than the top**

**then add the number and remove the top. DO the heapify**

- Repeat the process, at the end you will have Top 3 nums in an array

Same procedure can be done to find the smallest 3 numbers in an array

* Create a Max Heap of 3
* Add numbers one by one and compare with top element (Top element will be the Maximum number in the heap as it is a max heap)
* If the number you compared is lesser than the Top element then add to the heap or else do not add that number
* Repeat the process, at the end you will have 3 smallest num in array