**Polymorphism:**

Sometimes due to different conditions, we want our functions act differently. This is called polymorphism. Poly = many, Morphism = shapes (states)

**Polymorphism: Types**:

There are two types of Polymorphism. 1. Dynamic Polymorphism 2. Static Polymorphism

|  |  |
| --- | --- |
| **Dynamic Polymorphism** | **Static Polymorphism** |
| Function overriding is called Dynamic Polymorphism, because it decides at run time which function will be called. | Function overloading is form of a static polymorphism, because compiler decides at compile time which function will be called |
| In function overriding, the name and parameter list of the function should be same. | Function Overloading means same function name with different parameter list. |
| When we assign child class object to parent object and we want the functionality of parent object function is replaced with the child functionality. For this purpose, we add virtual keyword in the parent class and override keyword in child class. |  |

**Abstraction:**

* An abstract class is a class that is declared abstract.
* Abstraction means when something is an idea and not exist in concrete form (an event) in current context.
* Abstract classes cannot be instantiated, they can only be inherited.
* Abstract class instance can be created through it child class.
* When an abstract class is inherited, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.
* Abstraction applies more strict conditions on the child classes that they must give the definition of the abstract method or otherwise they become abstract as well.

|  |  |
| --- | --- |
| **Interface** | **Abstraction** |
| Interface support multiple inheritance | Abstract class does not support multiple inheritance |
| Interface does not contain data members | Abstract class can contain data members |
| Interface does not contain Constructors | Abstract class can contains Constructors |
| Interface only have abstract methods | Abstract class can have both concrete and abstract methods |
| Interface can not have access modifiers. By default everything is assumed public | Abstract class uses access modifiers to declare data members and member functions. |

**Association:**

**Association** has two types 1) Aggregation 2) Composition

* When life times of two separate objects are independent of each other, this type of association is called **Aggregation.**
* When life time of two separate objects are dependent of each other, this type of association is called **Composition**.

**Encapsulation:**

When we want to secure data for direct access from external world (objects of the class, other classes), this is called Encapsulation.

* We can make data member private, so that it can not be accessed directly from outside of the class.
* Access modifiers are used to implement Encapsulation of OOP. Access modifiers allow you to define who does or who doesn't have access to certain features.
* We can provide access to the data through the public getter and setter functions
* We can add constraints on the data inside the setter functions

**Inheritance:**

**Inheritance** allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors

* We create a general class that has common functionality. This class is also called the parent class (Base Class / Super Class).
* Then we create specific classes (Child Classes / Derived Classes / Sub Classes) that have different functionality to each other and inherit these classes from the common class.

**Constructor:**

Whenever we create an object of a class a constructor function is called automatically.

* Constructor function without any parameters is called default constructor.
* To create a default constructor, we use the same name as the class, followed by parentheses ()
* The constructor has the same name as the class, it is always public, and it does not have any return type.
* We can also pass multiple parameters to a constructor when an object is being created.
* Copy Constructor copies the data of one object into another object

**Classes and Objects:**

We can represent related data as a single unit that is called **class**.

* Class is a custom (user defined) data type that consists of attributes (sub component of any real time object).
* **Object** is a variable of class type that holds all information present in the class.
* We can use an object into any arithmetic and boolean expressions.
* We can create multiple objects of class to hold different information.
* Objects reference is created on Stack and the actual memory is allocated on Heap