**OOP CONCEPTS**

1. **Naming Conventions**
2. **Object and Class**
3. **Constructors and Destructors**
4. **Keyword “this”**
5. **Keyword “static”**
6. **Static Class and Static Constructor**
7. **Properties**
8. **INHERITANCE (IS-A)**
9. **AGGREGATION (HAS-A)**
10. **Member OVERLOADING**
11. **Methods OVERRIDING**
12. **Keyword “base”**
13. **POLYMORPHISM**
14. **Keyword “sealed”**
15. **Access Modifiers**
16. **ENCAPSULATION**
17. **Keyword “abstract”**
18. **Interfaces**
19. **Namespaces**

**Object Oriented Programming** is a paradigm that provides many concepts such as **DATA BINDING, ENCAPSULATION, INHERITANCE, POLYMORPHISM,** etc.

**OBJECT**

An entity that has STATE and BEHAVIOUR is known as an OBJECT (e.g. “chair”, “bike”, “marker”, “car”, etc.). It can be PHYSICAL or LOGICAL (tangible and intangible).

The example for an intangible object is BANKING SYSTEM.

An OBJECT has THREE characteristics:

* **STATE:** represents data (value) of an object
* **BEHAVIOUR:** represents the behavior (functionality) of an object
* **IDENTITY**

EXAMPLE: Pen is an OBJECT. Named “Reynolds”, color “black”, those represents its STATE. “Writing” is its BEHAVIOUR.

OBJECT:

* is a RUN TIME entity
* is an INSTANCE of a class
* is an entity which has STATE and BEHAVIOUR
* is a real world entity

**CLASS**

A **CLASS** is a **GROUP OF OBJECTS** which have **COMMON PROPERTIES**. It is a **TEMPLATE** or a **BLUEPRINT** from which objects are created. It is a **LOGICAL** entity, and it can not be physical.

A CLASS may contain:

* **FIELDS**
* **METHODS**
* **CONSTRUCTORS**
* **BLOCK**
* **NESTED CLASS AND INTERFACE**

A variable which is created inside a class, but outside a method, it is known as instance variable. An instance variable does not get memory at compile time, but at run time when an object (instance) is created.

Methods are used to define the behavior of an object. And their advantages are:

* **CODE REUSABILITY**
* **CODE OPTIMIZATION**

The “new” keyword is used to allocate memory at run time. All objects get their memory in HEAP memory area.

There are 3 ways to initialize an object:

* **by REFERENCE variable**
* **by METHOD**
* **by CONSTRUCTOR**

ANONYMOUS objects are simply nameless, they have no reference and can be used at the time of object creation only, as you can use them just once.

**CONSTRUCTORS**

**CONSTRUCTORS** are a special types of methods used to initialize objects and it is invoked at the time of creating the object.

**CONSTRUCTORS MUST:**

* HAVE THE SAME NAME AS THE CLASS
* HAVE NO EXPLICIT RETURN TYPE

**CONSTRUCTORS TYPE:**

* DEFAULT (if no constructor is defined, compiler provides one)
* PARAMETIZED

The DEFAULT CONSTRUCTOR provides the default values to the object like: 0, null, false, etc.

The PARAMETRIZED CONSTRUCTOR is used to initialize the object with certain values for its fields, and it can have any number of parameters. They can be OVERLOADED, by creating any number of constructors that DIFFER IN THE NUMBER OF PARAMETERS, OR THEIR TYPE.

|  |  |
| --- | --- |
| CONSTRUCTOR | METHOD |
| **Is used to initialize an object** | **Is used to expose the object’s behavior** |
| **Must not have a return type** | **Must have a return type** |
| **Is invoked implicitly** | **Is invoked explicitly** |
| **Complier provides a default constructor** | **Compiler does not provide a default method** |
| **Must have the same name as the class** | **Can have any name** |

**CONSTRUCTORS** return a value, and that is **THE CURRENT INSTANCE OF THE CLASS**. Can be used for creating objects, calling methods, starting threads, etc.

**“THIS”**

The keyword “this” is a reference variable that refers to the current instance of the class.

Usages of the keyword “this”:

* it can be used to refer current class instance variable
* to pass current object as an argument to a method call or to a constructor
* to declare indexers
* to invoke the current class method
* to return the current class instance from the method
* this( ) to invoke the current class constructor

If the parameter name has the same name as the field to be initialized it must be used the “this” keyword, otherwise it would be ambiguous for the compiler and it may result in an error or the constructor may initialize the fields with their default values as 0, null, false, etc.

**“STATIC”**

The keyword “static” is used mainly for memory management. It belongs to the type not to the instance. Can be used for:

* field
* method
* class
* properties
* operators
* events

So as “static” belongs to the type, it will not be allocated memory for static variables every time when an instance is created, but just one time, and is available for each instance. So is the common property of every instance.

* A static variable
* can be used to refer common properties of all objects
* get its memory allocation only one time at class loading time

So when you have to create more objects with the same property for all, make it static if you can, as it will save memory.

* A static method
* it belongs to the class rather than object of the class
* no need for creating an instance to invoke it
* can change static data (access it and modify its value)

The static methods can not use non static members or call non static methods directly. “This” and “base” can not be used in static context.

* A static class
* can contain only static members
* cannot be instantiated
* is sealed
* cannot contain instance constructors
* A static constructor
* cannot have any modifier or parameter
* is invoked implicitly, cannot be called
* used to initialize static fields

**Properties**

Properties doesn’t have storage location. They are extensions of fields and accessed as fields. Have accessors that are used to get or set the fields.

Usage of properties:

* can be read-only or write-only
* can have logic while setting values
* encapsulation, as we make fields private and can be accessed only via properties.

**INHERITANCE**

**(IS-A)**

Is the process in which one object acquires all the properties and behaviors of its parent object automatically. In such a way that you can reuse, extend or modify those.

**BASE CLASS -----> DERIVED CLASS**

It can be:

* SINGLE
* MULTILEVEL
* HIERARCHICAL

**AGGREGATION**

**(HAS-A)**

Is a process in which one class defines another class as any entity reference. It is another way to reuse the class. As it is a form of association that represents HAS-A relationship.

It is very good for code reusability. If the relation of inheritance is not maintained for ever between the classes, the AGGREGATION is better.

**OVERLOADING AND OVERRIDING**

OVERLOADING can be achieved by changing methods parameters numbers or their types.

OVERRIDING is the process where an object inherited the same behavior from its parents but achieve it different.

* same behavior, different implementation
* RUNTIME POLIMORPHISM

RULES for OVERIDING

* must have the same name
* must have the same parameters
* must be an IS-A relationship

**“BASE”**

The BASE keyword is used to access fields, constructors and methods of the base class.

CANOT be used in static context.

**POLYMORPHISM**

Many Forms.

* COMPILE TIME POLYMORPHISM
* RUN TIME POLYMORPHISM

COMPILE TIME POLYM.

* is achieved by using method and operator overloading, also known as static binding or early binding

RUN TIME POLYM.

* is achieved by using method overriding, also known as dynamic binding or late binding

**“SEALED”**

SEALED class cannot be derived, and sealed methods cannot be overridden.

**“ABSTRACT”**

ABSTRACT is the process of hiding internal details, showing only functionality.

* + - abstract methods cannot contain implementation
    - abstract methods can only be used in abstract classes
    - CANNOT BE INSTANTIATED
    - cannot have static and virtual keywords

**INTERFACES**

Are a blueprint for a class. Used to achieve multiple inheritance and FULLY ABSTRACTION.

The class which implements the interface must provide implementation for all the methods acquired from it.