1. 1- Abstract class

A class which has the abstract keyword in its declaration is called abstract class. Abstract classes should have zero or more abstract methods. i.e., methods without a body. It can have multiple concrete methods.

If you want to inherit the abstract class, you need to implement it first.

**Important Reasons for Using Abstract Class**

* Abstract classes offer default functionality for the subclasses.
* Provides a template for future specific classes.
* Helps you to define a common interface for its subclasses.
* Abstract class allows code reusability.

1. 2-Interface:

The interface is a blueprint that can be used to implement a class. The interface does not contain any concrete methods (methods that have code). All the methods of an interface are abstract methods.

An interface cannot be instantiated. However, classes that implement interfaces can be instantiated. Interfaces never contain instance variables but, they can contain public static final variables (i.e., constant class variables)

Comparison between interface and abstract classes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Interface | Abstract Class |  |  |
| Speed | Slow (adding methods requires code changes in all implementations) | Fast (adding methods only requires updating the interface) |  |  |
| Multiple Inheritances | Implements several interfaces | Only one abstract class |  |  |
| Structure | Abstract methods only | Both abstract and concrete methods |  |  |
| When to use | - Define shared method signatures for different implementations - Create polymorphic value types | - Share common behavior for different implementations - Provide partial implementation for concrete classes |  |  |
| Future enhancement | Difficult to update existing implementations with new methods | Can use default implementations in the abstract class for new methods |  |  |
| Access Modifiers | No access modifiers, all public | Can have access modifiers (public, private, protected) |  |  |
| Data fields | No data fields | Can have data fields |  |  |
| Multiple Inheritance Default | Multiple interfaces | Single abstract class |  |  |
| Implementation | Abstract, no code provided | Can provide complete or default code |  |  |
| Use of Access modifiers | Not available | Available within the class |  |  |
| Usage | Define "what" a class can do | Define "who" a class is (identity) |  |  |
| Defined fields | None | Allowed |  |  |
| Inheritance | Inherits multiple interfaces | Inherits one class or one abstract class |  |  |
| Constructor or destructors | No constructor or destructor | Can declare constructors and destructors |  |  |
| Limit of Extensions | Extends any number of interfaces | Extends only one class or one abstract class at a time |  |  |
| Abstract keyword | Optional for declaring abstract methods | Required for declaring abstract methods |  |  |
| Class type | Only public abstract methods | Public and protected abstract methods |  |  |

1. inheriting a previously defined constructor function means using the parameters of the previously defined function along with adding some new parameters to the newly defined constructor function. For this, we need to use the call() function which allows us to call a function defined somewhere else in the current context.

Lets say for example we have a human object :

Function Human(fname,lname){

This.fname=fname;

This.lname=lname;

}

Then we have a developer who is a human too and has some extra method so to apply the DRY principles we are inheriting the human object.

Function Dev(fname,lname,specialization){

Human.call(this, fname, lname)

This.specialization=specialization;

}

We can notice that the Dev constructor function inherits the properties of the ‘Human‘ constructor function along with a new parameter ‘specialization‘. Here, we called the Employee function using the call() function to pass the required parameters to the Employee constructor function.

If we logged the answers console.log (Human. Prototype) constructor function: Human(fname,lname)

Will be the output.

console.log(Dev.prototype):

constructor function: Human(fname, lname, specialization).