Classes vs Abstract Classes vs Interfaces

Classes:

TypeScript is object oriented JavaScript. TypeScript supports object-oriented programming features like classes, interfaces, etc. A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object. Typescript gives built in support for this concept called class. JavaScript ES5 or earlier didn’t support classes. Typescript gets this feature from ES6.

The class keyword is followed by the class name. The rules for identifiers must be considered while naming a class.
A class definition can include the following :

- Fields: A field is any variable declared in a class. Fields represent data pertaining to objects
- Constructors: Responsible for allocating memory for the objects of the class
- Functions: Functions represent actions an object can take. They are also at times referred to as methods

```
class Car {
  //field
  engine:string;
  //constructor
  constructor(engine:string) {
    this.engine = engine
  }
  //function
  disp():void {
    console.log("Engine is  :   "+this.engine)
  }
}
```

Abstract Classes:

An abstract class is a class that includes both abstract and regular methods. It is a class that is inherited by multiple classes. We cannot create objects of an abstract class.
An abstract class is declared by using the keyword abstract.

- An abstract class generally has one or more abstract methods.
- An abstract method is one that doesn't have any implementation. It merely defines the method's signature and excludes the method body. In the derived class, an abstract method must be implemented.

```
abstract class Animal {
   public constructor() {
      console.log('animal init');
   }
}
class Dog extends Animal {
   public makeSound(): void {
      console.log('bark');
   }
}
const dog = new Dog();
```

Interfaces:

Interface is a structure that defines the contract in your application. It defines the syntax for classes to follow. Classes that are derived from an interface must follow the structure provided by their interface. The TypeScript compiler does not convert interface to JavaScript. It uses interface for type checking. This is also known as "duck typing" or "structural subtyping". An interface is defined with the keyword interface, and it can include properties and method declarations using a function or an arrow function.

```
interface KeyPair {
   key: number;
   value: string;
}
let kv1: KeyPair = { key:1, value:"Steve" }; // OK
let kv2: KeyPair = { key:1, val:"Steve" }; // Compiler Error: 'val' doesn't exist in type 'KeyPair'
let kv3: KeyPair = { key:1, value:100 }; // Compiler Error:
```

References:

https://www.codingninjas.com/codestudio/library/abstract-classes-in-typescript

https://www.tutorialsteacher.com/typescript/typescript-interface

https://www.tutorialspoint.com/typescript/typescript_classes.htm