Static Methods

Similar to *static* variables, ***static* methods also belong to a class** instead of the object, and so they can be **called without creating the object of the class** in which they reside. They're meant to be used without creating objects of the class.

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

***static* methods** are generally used to perform an operation that is **not dependent upon instance creation**.

If there is a code that is supposed to be **shared across all instances of that class**, then write that code in a *static* method.

A static method can **access only static variables of class** and **invoke only static methods of the class**

The **main() method** that is the entry point of a java program itself **is a static method**. Wonder why? It is **because the object is not required to call a static method**. If it were a non-static method, **JVM creates an object first then call main() method that will lead the problem of extra memory allocation**.

static methods are also widely used to **create utility or helper classes** so that they can be obtained without creating a new object of these classes.

Why to use static methods???

- To access/manipulate static variables and other static methods that don't depend upon objects.
- *static* methods are widely used in utility and helper classes.

Points to be remembered…

- *static* methods in Java are resolved at compile time. Since method overriding is part of Runtime Polymorphism, **so static methods can't be overridden.**
- abstract methods can't be static. //will be covered in future classes
- *static* methods **cannot use *this* or *super* keywords**. //will be covered in future classes
- The static method **cannot use non static data member** or call non-static method directly.
- The following combinations of the instance, class methods and variables are valid:
    1. Instance methods can directly access both instance methods and instance variables
    2. Instance methods can also access *static* variables and *static* methods directly
    3. *static* methods can access all *static* variables and other *static* methods
    4. ***static* methods cannot access instance variables and instance methods directly**; they need some object reference to do so

Now let's start with new concept which is one of the important features of OOP which is INHERITANCE

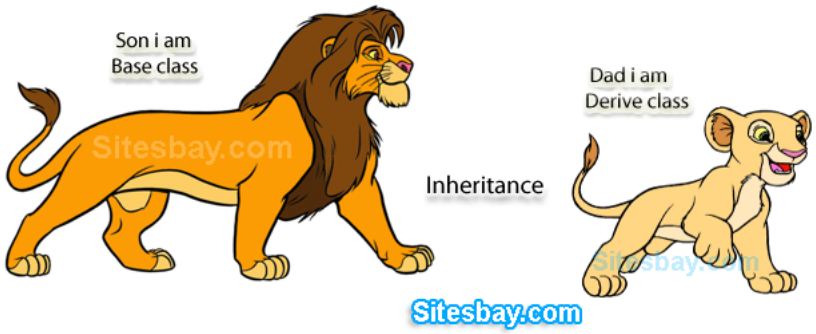
Inheritance in Java is a mechanism in which **one object acquires all the properties and behaviours of a parent object**. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can **create new classes that are built upon existing classes**. When you inherit from an existing class, **you can reuse methods and fields of the parent class**. Moreover**, you can add new methods and fields in your current class also**.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Child Class

The class that **extends the features of another class** is known as child class, sub class or derived class.



Parent Class

The **class whose properties and functionalities are used (inherited) by another class** is known as parent class, super class or Base class.

- One of the key benefits of inheritance is to **minimize the amount of duplicate code in an application by sharing common code amongst several subclasses**. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilation units.
- **Inheritance can also make application code more flexible** to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass
- **Reusability** - facility to use public methods of base class without rewriting the same.
- **Extensibility** - extending the base class logic as per business logic of the derived class.
- **Data hiding** - base class can decide to keep some data private so that it cannot be altered by the derived class
- **Overriding** -With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

The syntax of Java Inheritance

```
class subclass_name extends superclass_name
{
    //body of the class
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Let's understand with code

```
class CreditCard  // parent class
{
    int cardNo = 12345;
    int pin = 8888;
}
class Hacker extends CreditCard  //child class
{
    void viewDetails()
    {
        System.out.println(cardNo); //inherited variables
        System.out.println(pin);
    }
    void changeDetails()
    {
        cardNo = 6789; // Overriding variables
        pin = 9999;
    }
}
class Demo
{
    public static void main(String[] args)
    {
        Hacker h = new Hacker();
        h.viewDetails();
        h.changeDetails();
        h.viewDetails();
    }
}
```

Output

```
12345
8888
6789
9999
```

There are certain rules in inheritance in java

- **Rule1:** Private members do not participate in inheritance, this is so that encapsulation does not get affected. However public, protected and default members participate in inheritance. (You'll learn about private, public, default and protected in access specifiers in future classes)

// Other rules will be covered in future classes.