Inheritance is a fundamental concept in object-oriented programming (OOP) that allows classes to inherit properties and behaviors from other classes. In Java, inheritance is implemented using the "extends" keyword, where one class (the subclass) extends another class (the superclass).

Here are some important details about inheritance in Java:

1. Inheritance allows the subclass to inherit all the non-private fields and methods of the superclass. Private fields and methods are not inherited and are only accessible within the class where they are defined.
2. A subclass can only extend one superclass, but a superclass can have multiple subclasses.
3. The subclass can override the methods of the superclass, which means it can provide its own implementation of a method with the same name and parameters as the superclass method. The subclass can also call the overridden method of the superclass using the "super" keyword.
4. The subclass can access the public and protected fields and methods of the superclass directly, but it cannot access the private fields and methods.
5. The subclass can add its own fields and methods, which are not present in the superclass. These fields and methods are only accessible within the subclass.
6. The constructor of the superclass is not inherited by the subclass, but the subclass can call the constructor of the superclass using the "super" keyword.
7. Inheritance creates an "is-a" relationship between the subclass and the superclass. For example, if a class called "IndianPerson" extends a class called "Person", we can say that "IndianPerson" is a type of "Person".

In summary, inheritance is a powerful feature of Java that allows classes to reuse and extend the functionality of other classes. It promotes code reusability and makes the code more modular and easier to maintain.

Some more details on **inheritance**:

1. **Single Inheritance:** Java supports single inheritance, which means that a subclass can only extend one superclass. This is in contrast to some other languages, like C++, which support multiple inheritance. However, Java does support the use of interfaces, which can be implemented by a class in addition to its superclass. Interfaces define a set of methods that a class must implement, but do not provide any implementation themselves.
2. **Method Overriding:** When a subclass defines a method with the same name, return type, and parameters as a method in the superclass, it is said to be "overriding" the superclass method. The subclass method can provide its own implementation of the method, which will be used instead of the superclass implementation when the method is called on an instance of the subclass. Method overriding is a powerful tool for creating more specific behavior in a subclass while still reusing code from the superclass.

3. **Access Modifiers:** In Java, access modifiers (public, protected, private) control the visibility of fields and methods. When a field or method is declared as public or protected in a superclass, it can be accessed by the subclass using dot notation (e.g. superclass.fieldName or superclass.methodName()). Private fields and methods are not visible to the subclass. It's worth noting that if a subclass overrides a method from the superclass, it can choose to make the overriding method more visible (e.g. changing a protected method in the superclass to public in the subclass), but it cannot make it less visible.
4. **Adding Fields and Methods:** In addition to inheriting fields and methods from the superclass, a subclass can add its own fields and methods. These fields and methods are only visible within the subclass and are not accessible by code outside the subclass.
5. **Super Keyword:** The "super" keyword in Java is used to refer to the superclass from within the subclass. It can be used to call the superclass constructor, access fields and methods of the superclass, and call the superclass implementation of an overridden method.
6. **"is-a" Relationship:** Inheritance in Java creates an "is-a" relationship between the subclass and the superclass. This means that a subclass is a type of the superclass. For example, if a class called "Cat" extends a class called "Animal", we can say that "Cat" is a type of "Animal". This "is-a" relationship is important for understanding how inheritance works in Java and for designing effective class hierarchies.

Example of inheritance:

Parent person class:

```
package parent;

import child.USPerson;

public class Person {

   protected String name = null;
   protected int age = 0;

   public Person(int a, String n) {
      this.age = a;
      this.name = n;
   }

   public Person(String b1) {
      this.name = b1;
   }

   public void personDetails() {
      System.out.println("parent.Person Name is " + name + " and age" + age);
```

 Contact: 8087883669

```
    }

}
```

First child of Person class:

```
package child;

import parent.Person;

public class IndianPerson extends Person {
    String aadharCardNumber;
    public IndianPerson(int age1, String name, String aadharCardNumber){
        super(age1, name);
        this.aadharCardNumber = aadharCardNumber;
    }

    @Override
    public void personDetails(){
        //Super is calling method of parent
        super.personDetails();
        System.out.println("And aadhar number is "+aadharCardNumber);
    }

}
```

Second child of Person class:

```
package child;

import parent.Person;

public class IndianPerson extends Person {
    String aadharCardNumber;
    public IndianPerson(int age1, String name, String aadharCardNumber){
        super(age1, name);
        this.aadharCardNumber = aadharCardNumber;
    }

    @Override
    public void personDetails(){
        //Super is calling method of parent
        super.personDetails();
        System.out.println("And aadhar number is "+aadharCardNumber);
    }
```



```
}
```

Main runner class :

```
import child.IndianPerson;
import child.USPerson;
import parent.Person;

public class PersonMain {
   public static void main(String[] args) {

      //Creating object of parent class
      Person personobject= new Person(25,"Sourabh");
      personobject.personDetails();

      Person indianPerson = new IndianPerson(25,"Sourabh", "2125442251");
      indianPerson.personDetails();

      Person usPerson = new USPerson(25,"Sourabh", "2125442251");
      usPerson.personDetails();

      //also parent data type can hold reference of parent as well as child
      Person indianPerson1 = new IndianPerson(25,"Sourabh", "2125442251");
      indianPerson1.personDetails();

      Person usPerson1 = new USPerson(25,"Sourabh", "2125442251");
      usPerson1.personDetails();


   }
}
```

Assignment question:

Create Vehicle class with name, and vehicle registration number
Create method in Vehicle class to print name and registration number

Create child class, Truck, Bus
Truck will have additional field called load capacity
Bus will have additional field called number of passanger.

Create method in child classes to print all, name, registration number and child specific details.