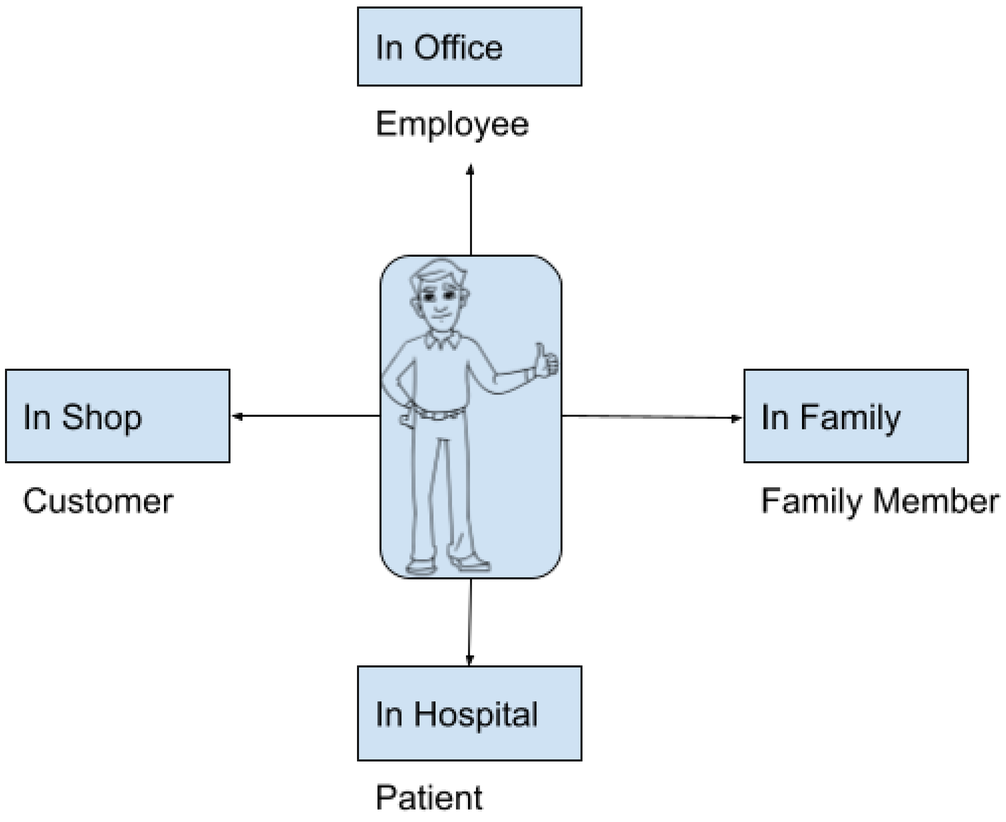
Polymorphism
============

It is one of the OOPs concepts, where one object performs different behaviour at different intervals of time.

E.x. Here we can see in real life, one person can perform different behaviour at different interval of time.



Types of polymorphism
—-----------------------------

a. **Compile time polymorphism**

- In compile time polymorphism, method declaration is going to get binded with method definition(method body) at the time of compilation based on argument.
- It is also called early binding.
- Ex. method overloading

b. **Run time polymorphism**

- In run time polymorphism, method declaration is going to get binded with method definition(method body) at the time of execution based on object creation.
- It is also called late binding.
- Ex. method overriding

Difference between method overloading and overriding

Method overloading	Method overriding
1. Same name of two or more than two methods in the same class. 2. Inheritance is not required.	1. Same name of two or more than two methods but in different classes. 2. Inheritance must be required.
3. Different arguments - Number of arguments - Sequence of arguments - Types of arguments	3. Same arguments - Number of arguments - Sequence of arguments - Types of arguments

1. Method overloading

- Method overloading is declaring multiple methods with the same name but different arguments.

Ex.

```
—-------------------------------------
public class aclass
{
        public void test1()                          // method with zero argument
        {
                System.out.println("property of method test1");
        }

        public void test1(int i)                     // method with single argument
        {
                System.out.println("property of method test2");
        }

        public void test1(int i, String s)           // method with two argument
        {
                System.out.println("property of method test3");
        }

        public static void main(String[] args)
        {
                aclass a = new aclass();
                a.test1();                           // calling method having zero argument
                a.test1(1);                          // calling method having single argument
                a.test1(1,"amit");                   // calling method having two argument
        }
}
```

Program Output :

```
property of method test1
property of method test2
property of method test3
```

2. **Method overriding**

- Method overriding is acquiring super class methods into subclass and changing the implementation of methods with respect to subclass specifications.

E.x

a. **Method with same method name with same sequence of arguments in different classes.**

—---

Super class >>

```
public class Superclass
{
	public void test1(int i, String s)				// superclass property
	{
		System.out.println("Super class property");
	}
}
```

Subclass >>

```
public class Subclass extends Superclass
{
	public void test1(int i, String s)				// subclass property
	{
		System.out.println("Subclass property");
	}

	public static void main(String[] args)
	{
	Subclass s = new Subclass();		// creating object of subclass
	s.test1(1,"amit"); // calling superclass method but changing the implementation
according subclass
	}
}
```

Program Output : Subclass property

—---

E.x

b. **Method with same method name with same type of arguments in different classes.**

—--

Super class >>

```
public class Superclass
{
	public void test1(int a)				// superclass property
	{
		System.out.println("Super class property");
	}
}
```

Subclass >>

```
public class Subclass extends Superclass
{
	public void test1(int a)				// subclass property
	{
		System.out.println("Subclass property");
	}
	public static void main(String[] args)
	{
		Subclass s = new Subclass();		// creating object of subclass
		s.test1(1); // calling superclass method but changing the implementation
according subclass
	}
}
```

Program Output : Subclass property

—--

E.x

c. **Method with same method name with same numbers of arguments in different classes.**

—--

Super class >>

```
public class Superclass
{
	public void test1(int a, int b)			// superclass property
	{
		System.out.println("Super class property");
	}
}
```

Subclass >>

```
public class Subclass extends Superclass
{
	public void test1(int a, int b)				// subclass property
	{
		System.out.println("Subclass property");
	}
	public static void main(String[] args)
	{
		Subclass s = new Subclass();		// creating object of subclass
		s.test1(1,2); // calling superclass method but changing the implementation
according subclass

	}
}
```

Program Output : Subclass property