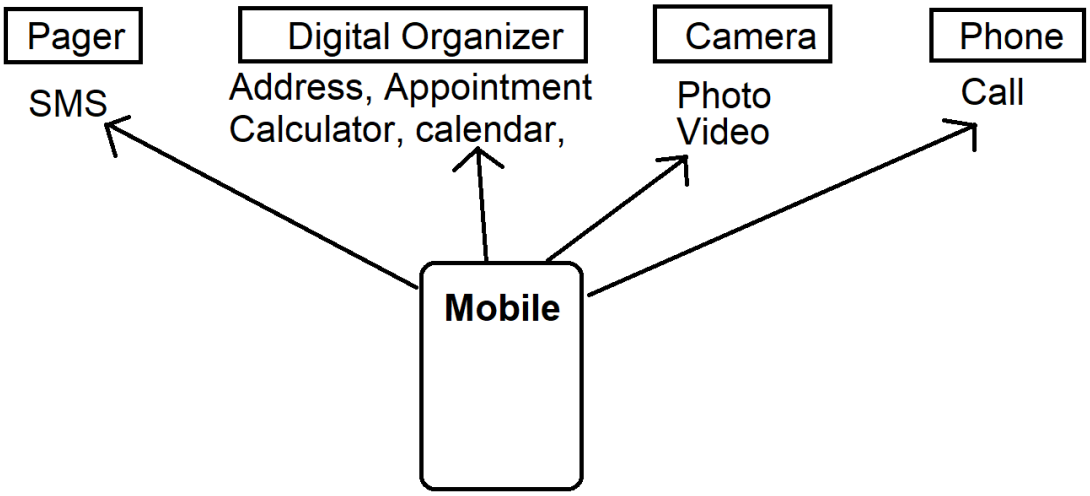
Polymorphism

- Poly means many
- Morphos means Forms



Poly Morphism

- It is a technique used to define multiple behaviours to single component.
- At various instances a component can perform different tasks.
- According to state and situation a component can change its functionality.
- If single component is allocated with different types of memories then it can exhibit Polymorphism.
- **A single base class object can use the memory of multiple derived classes exhibits Polymorphism.**

Ex:

```
class Employee
{
   public firstName:string;
   public lastName:string;
```

```
    public designation:string;
    public Print():void {
        console.log(`${this.firstName} ${this.lastName} -
${this.designation}`)
    }
}
class Developer extends Employee
{
    public firstName = "Kiran";
    public lastName = "Kumar";
    public designation = "Developer";
    public Print(){
        super.Print();
        console.log("Developer Role: Design, Build, Debug,
Test");
    }
}
class Admin extends Employee
{
    public firstName = "Raj";
    public lastName = "Kumar";
    public designation = "Admin";
```

```
    public Print(){
        super.Print();
        console.log("Admin Role: Credentials, Configurations,
Mapping");
    }
}
class Manager extends Employee
{
    public firstName = "Tom";
    public lastName = "Hanks";
    public designation = "Manager";
    public Print(){
        super.Print();
        console.log("Manager Role: Approvals");
    }
}
let employees:Employee[] = new Array(new Developer(),
new Admin(), new Manager());
for(var employee of employees)
{
    employee.Print();
}
```

Ex:

```
class Employee
{
   public firstName:string;
   public lastName:string;
   public designation:string;
   public Print():void {
      console.log(`${this.firstName} ${this.lastName} -
${this.designation}`)
   }
}
class Developer extends Employee
{
   public firstName = "Kiran";
   public lastName = "Kumar";
   public designation = "Developer";
   public Print(){
      super.Print();
      console.log("Developer Role: Design, Build, Debug,
Test");
   }
}
```

```
class Admin extends Employee
{
   public firstName = "Raj";
   public lastName = "Kumar";
   public designation = "Admin";
   public Print(){
      super.Print();
      console.log("Admin Role: Credentials, Configurations,
Mapping");
   }
}
class Manager extends Employee
{
   public firstName = "Tom";
   public lastName = "Hanks";
   public designation = "Manager";
   public Print(){
      super.Print();
      console.log("Manager Role: Approvals");
   }
}
let employees:Employee[] = [];
```

```
employees[0] = new Developer();
employees[1] = new Admin();
employees[2] = new Manager();

for(var i=0; i<employees.length; i++){
   employees[i].Print();
}
```

Ex:

- **Create a new TS file “demo.ts”**

```
class Employee
{
   public firstName:string;
   public lastName:string;
   public designation:string;
   public Print():void {
      document.write(`${this.firstName} ${this.lastName}
- ${this.designation}<br>`)
   }
}
class Developer extends Employee
{
   public firstName = "Kiran";
   public lastName = "Kumar";
   public designation = "Developer";
   public Print(){
      super.Print();
```

```
        document.write("Developer Role: Design, Build,
Debug, Test<br>");
    }
}
class Admin extends Employee
{
    public firstName = "Raj";
    public lastName = "Kumar";
    public designation = "Admin";
    public Print(){
        super.Print();
        document.write("Admin Role: Credentials,
Configurations, Mapping<br>");
    }
}
class Manager extends Employee
{
    public firstName = "Tom";
    public lastName = "Hanks";
    public designation = "Manager";
    public Print(){
        super.Print();
        document.write("Manager Role: Approvals<br>");
    }
}
let employees:Employee[] = [];
employees[0] = new Developer();
employees[1] = new Admin();
employees[2] = new Manager();
```

- **Trans compile into JS [** > tsc demo.ts]
- **Add a new HTML file into project “**home.html”

```
<!DOCTYPE html>
<html>
   <head>
      <script src="demo.js"></script>
      <script>
         function bodyload(){
          var designation = prompt("Enter Designation");
          for(var i=0; i<employees.length; i++)
          {
             if(employees[i].designation==designation)
             {
                employees[i].Print();
             }
          }
         }
      </script>
   </head>
   <body onload="bodyload()">
   </body>
</html>
```

TASK:

- Create a super class
- With Print() Method
- Design Print method for 3 different operations like “Addition, sub, multiplication”
- Obj.Print(add, a, b) // addition

- Obj.Print(mul, a, b) // mul
- Obj.Print(sub, a, b) // subtraction
- UI in HTML
  Prompt(operation)? Add
  Prompt(a)
  Prompt(b)