Inheritance in Kotlin

Inheritance is a feature of OOP that enables us to create a new class from an existing one thereby extending its functionality or reducing code duplication.

The child class inherits all the features of the parent class and can also have its own features.

Consider the following 3 classes:

```
class Banker(name: String, age: Int){
 fun talk(words: String){
 println(words)
 }

 fun eat(){
 println("yum")
 }

 fun sleep(){
 println("zzzzz")
 }

 fun countMoney(notes: List<Int>): Int{
 var sum = 0
 notes.forEach{note-> sum+=note}
 return sum
 }
}

class Doctor(name: String, age: Int){
 fun talk(words: String){
 println(words)
 }

 fun eat(){
 println("yum")
 }

 fun sleep(){
 println("zzzzz")
 }

 fun treatPatient(patient: String, disease: String){
 println("Treat $patient for $disease")
 }
}

class Farmer(name: String, age: Int){
 fun talk(words: String){
 println(words)
 }
 fun eat(){
 println("yum")
```

```
}

fun sleep(){
println("zzzzz")
}

fun cultivateLand(){
println("dig dig dig")
}
}
```

Each of the three classes has some duplicated methods. Instead of such repetition we can do this instead

```
open class Person(name: String, age: Int){
 fun talk(words: String){
 println(words)
 }

 fun eat(){
 println("yum")
 }

 fun sleep(){
 println("zzzzz")
 }
}

class Banker(name: String, age: Int): Person(name, age){
 fun countMoney(notes: List<Int>): Int{
 var sum = 0
 notes.forEach{note-> sum+=note}
 return sum
 }
}

class Doctor(name: String, age: Int): Person(name, age){
 fun treatPatient(patient: String, disease: String){
 println("Treat $patient for $disease")
 }
}

class Farmer(name: String, age: Int): Person(name, age){
 fun cultivateLand(){
 println("dig dig dig")
 }
}
```

Each of the 3 classes now inherits the Person class and each of them can access the methods in the parent class e.g:

```
fun main(){
 var aloo = Farmer("Aloo", 21)
 aloo.sleep() //zzzzz
 aloo.cultivateLand() //dig dig dig
}
```

Overriding Parent Class Functions

Under certain circumstances we may want an inherited function to behave differently in the child class as opposed to the parent class definition. In such cases we can override the parent function definition like so:

```
open class Person(name: String, age: Int){
 fun talk(words: String){
 println(words)
 }

 open fun eat(){
 println("yum")
 }

 fun sleep(){
 println("zzzzz")
 }
}

class Farmer(name: String, age: Int): Person(name, age){
 fun cultivateLand(){
 println("dig dig dig")
 }

 override fun eat(){
 println("I am eating all the food that I have grown")  }
}

fun main(){
 var aloo = Farmer("Aloo", 21)
 aloo.eat() //I am eating all the food that I have grown }
```

We override the `eat()` function by marking in as open in the parent class then redeclaring it in the child class with the `override` keyword. In order to inherit from a class we must mark it as open as well.

Calling parent class functions from child class

When we override a function we can call the parent class implementation using the `super` keyword

```
override fun eat(){
 super.eat()
 println("I am eating all the food that I have grown")
}
```

This outputs:

```
yum
I am eating all the food that I have grown
```