Mobile Application Development

Week3　The Object-oriented Programming of Kotlin

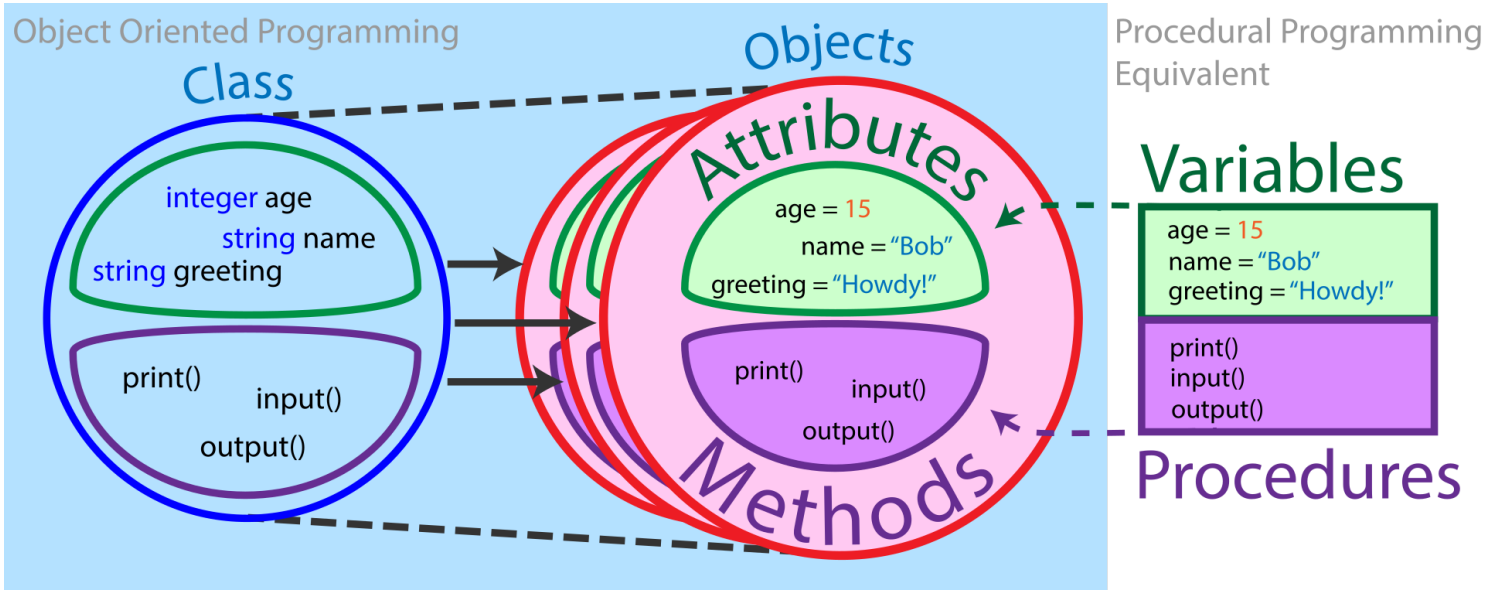
Yi SUN
Kobe Institute of Computing

The Object-oriented Programming of Kotlin

Class and Instance

Object-oriented Programming

- Write CLASSES to represent real-world.
- Create objects base on these classes = instances
- In Kotolin
  - Attribute = Property
  - Method = function



https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Fundamentals_of_programming/Elements_of_Object-Oriented_Programming

https://kotlinlang.org/docs/classes.html

Creating a Class

class name

```
class Dog(val name: String, val age: Int) {

    fun sit() {
        println("¥$name is now sitting.")
    }

    fun rollOver() {
        println("¥$name rolled over!")
    }
}
```

define a sit() function for the Dog Class

define a roll_over() function for the Dog Class

Making an Instance from a Class

example

```
fun main() {
    val myDog = Dog("Willie", 6)

    println("My dog's name is ${myDog.name}.")
    println("My dog is ${myDog.age} years old.")
}
```

create an instance of Dog class

result

My dog's name is Willie.
My dog is 3 years old.

Calling functions

example

```
fun main() {
    val myDog = Dog("Willie", 6)

    myDog.sit()
    myDog.rollOver()
}
```

result

Willie is now sitting.
Willie rolled over!

Creating Multiple Instances

example

```
fun main() {
    val myDog = Dog("Willie", 6)
    val yourDog = Dog("Lucy", 3)

    println("My dog's name is ${myDog.name}.")
    println("My dog is ${myDog.age} years old.")
    myDog.sit()

    println("Your dog's name is ${yourDog.name}.")
    println("Your dog is ${yourDog.age} years old.")
    yourDog.sit()
}
```

result

My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.
Your dog's name is Lucy.
Your dog is 3 years old.
Lucy is now sitting.

Working with Classes and Instances

example: Car class

```
class Car(val make: String, val model: String, val year: Int) {
    fun getDescriptiveName(): String {
        return "$year $make $model"
    }
}

fun main() {
    val myNewCar = Car("Audi", "A4", 2019)
    println(myNewCar.getDescriptiveName())
}
```

Setting a Default Value for a property

```
class Car(val make: String, val model: String, val year: Int) {
    var odometerReading: Int = 0

    fun getDescriptiveName(): String {
        return "$year $make $model"
    }

    fun readOdometer() {
        println("This car has $odometerReading miles on it.")
    }
}

fun main() {
    val myNewCar = Car("Audi", "A4", 2019)
    println(myNewCar.getDescriptiveName())
    myNewCar.readOdometer()
}
```

Add a property in the Car class with a default value

A new function to access the odometer data

Modifying a property's Value Directly

example

```
fun main() {
    val myNewCar = Car("Audi", "A4", 2019)
    println(myNewCar.getDescriptiveName())
    myNewCar.odometerReading = 23
    myNewCar.readOdometer()
}
```

result

```
2019 Audi A4
This car has 23 miles on it.
```

Modifying a property's Value Through a function

example

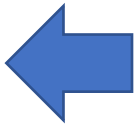
```
class Car(val make: String, val model: String, val year: Int) {
    var odometerReading: Int = 0

    fun getDescriptiveName(): String {
        return "$year $make $model"
    }

    fun readOdometer() {
        println("This car has $odometerReading miles on it.")
    }

    fun updateOdometer(mileage: Int) {
        odometerReading = mileage
    }
}

fun main() {
    val myNewCar = Car("Audi", "A4", 2019)
    myNewCar.odometerReading = 23
    myNewCar.readOdometer()
    myNewCar.updateOdometer(100)
    myNewCar.readOdometer()
}
```

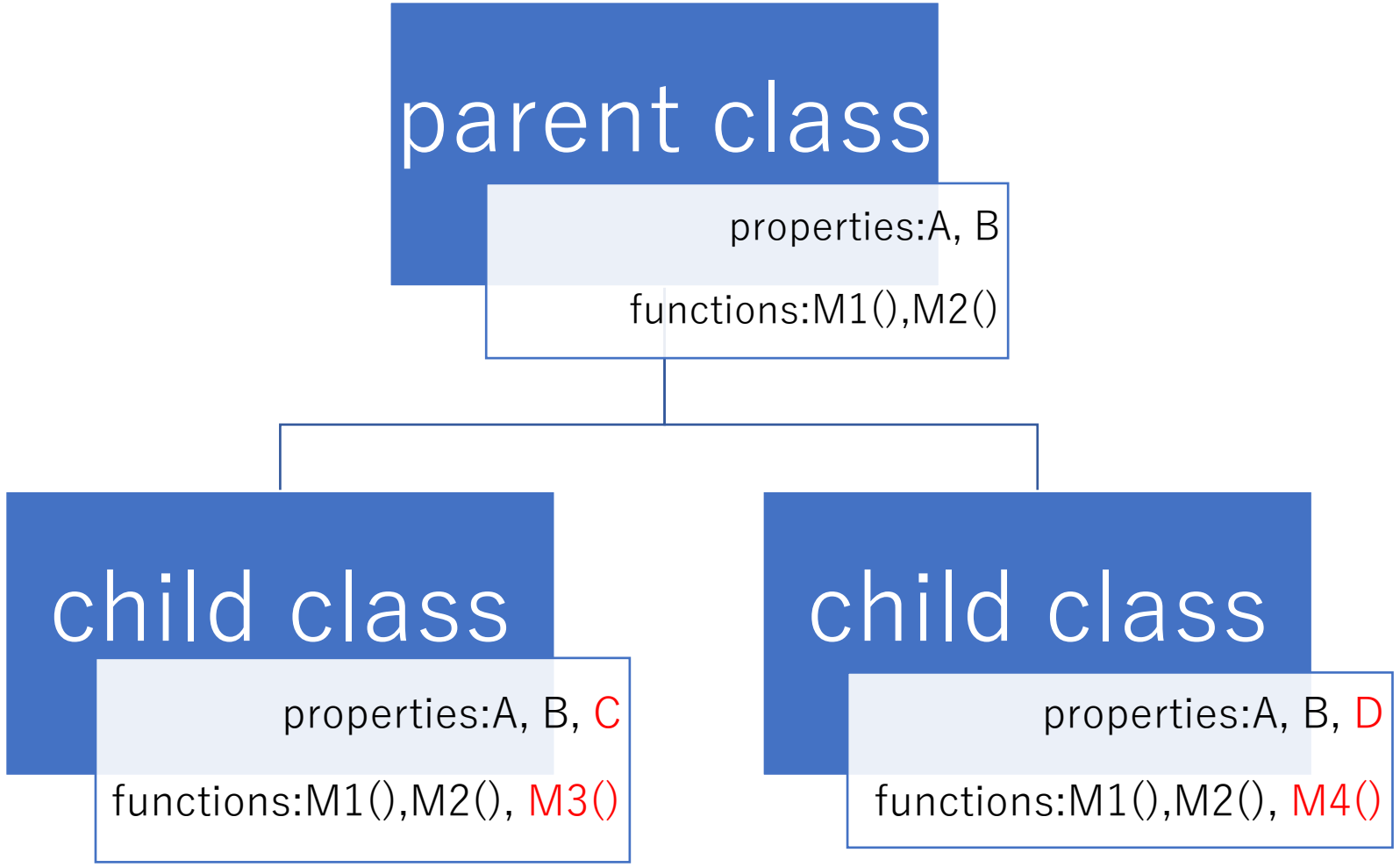


A new function to update the data of object

result

This car has 23 miles on it.
This car has 100 miles on it.

Inheritance



Inheritance in Kotlin

example

parent class

```
class ElectricCar(make: String, model: String, year: Int) : Car(make, model, year) {
    // No additional properties or functions for now
}

fun main() {
    val myTesla = ElectricCar("Tesla", "Model S", 2019)
    println(myTesla.getDescriptiveName())
}
```

In Kotlin, classes are **final by default**, which means they **cannot be inherited** unless explicitly marked as open To allow inheritance, modify the **Car class** as follows:

```
open class Car(val make: String, val model: String, val year: Int) {
        …
}
```

Defining Attributes and Methods for the Child Class

example

```
class ElectricCar(make: String, model: String, year: Int) : Car(make, model, year) {
    private var batterySize: Int = 75

    fun describeBattery() {
        println("This car has a $batterySize-kWh battery.")
    }
}

fun main() {
    val myTesla = ElectricCar("Tesla", "Model S", 2019)
    println(myTesla.getDescriptiveName())
    myTesla.describeBattery()
}
```

The original property of ElectricCar class

The original function() of ElectricCar class

result

2019 Tesla Model S
This car has a 75-kWh battery.

Overriding functions from the Parent Class

example

Add a new open function fillGasTank in Car class file

```
open fun fillGasTank() {
    println("Filling the gas tank.")
}
```

```
class ElectricCar(make: String, model: String, year: Int) : Car(make, model, year) {
    private var batterySize: Int = 75

    fun describeBattery() {
        println("This car has a $batterySize-kWh battery.")
    }

    override fun fillGasTank() {
        println("This car doesn't need a gas tank!")
    }
}
```

overriding the fill_gas_tank() method

result

2019 Tesla Model S
This car has a 75-kWh battery.
This car doesn't need a gas tank!

Instances as property

example

```
open class Car(val make: String, val model: String, val year: Int) {
    var odometerReading: Int = 0

    fun getDescriptiveName(): String {
        return "$year $make $model".replaceFirstChar { it.uppercase() }
    }
}

class Battery(private val batterySize: Int = 75) {
    fun describeBattery() {
        println("This car has a $batterySize-kWh battery.")
    }
}

class ElectricCar(make: String, model: String, year: Int) : Car(make, model, year) {
    val battery = Battery()
}

fun main() {
    val myTesla = ElectricCar("Tesla", "Model S", 2019)
    println(myTesla.getDescriptiveName())
    myTesla.battery.describeBattery()
}
```

Data Class

Special classes designed to hold data with automatically generated utility functions.

```
data class Person(val name: String, val age: Int, val city: String)

fun main() {
    // Creating instances
    val person1 = Person("Alice", 25, "Tokyo")
    val person2 = Person("Alice", 25, "Tokyo")
    val person3 = Person("Bob", 30, "Kobe")

    // 1 equals() - Checks if two objects have the same property values
    println("person1 equals person2: ${person1 == person2}")  // true
    println("person1 equals person3: ${person1 == person3}")  // false

    // 2 hashCode() - Generates a unique hash code based on property values
    println("person1 hashCode: ${person1.hashCode()}")
    println("person2 hashCode: ${person2.hashCode()}")  // Same as person1
    println("person3 hashCode: ${person3.hashCode()}")  // Different

    // 3 toString() - Returns a string representation
    println("person1 toString(): $person1")
    // Output: Person(name=Alice, age=25, city=Tokyo)

    // 4 copy() - Creates a new object with optionally modified properties
    val person4 = person1.copy(age = 26, city = “Kampala")
    println("person4 (copied with new age and city): $person4")
    // Output: Person(name=Alice, age=26, city=Kampala)

    // 5 componentN() - Used for destructuring
    val (name, age, city) = person1
    println("Destructured values: Name=$name, Age=$age, City=$city")
}
```

Kotlin Programming Practice

Bank Account Management System

Objective

- Implement a simple **Bank Account Management System** using **Kotlin**.
- Understand **class creation,property, and functions** in Kotlin.

Task Overview

You are required to implement a **BankAccount** class that supports:

1.Deposits

2.Withdrawals

3.Transaction History Logging

4.Displaying Transaction History

After implementing the class, write a **main function** that demonstrates its functionality.

Requirements

1. BankAccount Class

•The class should contain the follow properties:

- accountHolder: The name of the account holder.
- balance: The account balance.
- A **private list** to store transaction history.

2.Functions

- deposit(amount: Double): Adds money to the balance and logs the transaction.
- withdraw(amount: Double): Deducts money from the balance if funds are sufficient, otherwise prints an error.
- displayTransactionHistory(): Prints all transactions made.

Code Example

Your implementation should match the following expected behavior:

```
fun main(){
    val sunyiBankAccount = BankAccount("Yi Sun", 100.00)
    sunyiBankAccount.deposit(10.00)
    println(sunyiBankAccount.balance)
    sunyiBankAccount.withdraw(80.00)
    println(sunyiBankAccount.balance)
    sunyiBankAccount.deposit(500.0)
    sunyiBankAccount.withdraw(1300.0)
    sunyiBankAccount.displayTransactionHistory()
}
```

Expected Output

```
110.0
30.0
You don't have the funds to withdraw $1300.0 Transcation history for Yi Sun
Yi Sun deposited $10.0
Yi Sun withdrew $80.0
Yi Sun deposited $500.0
```