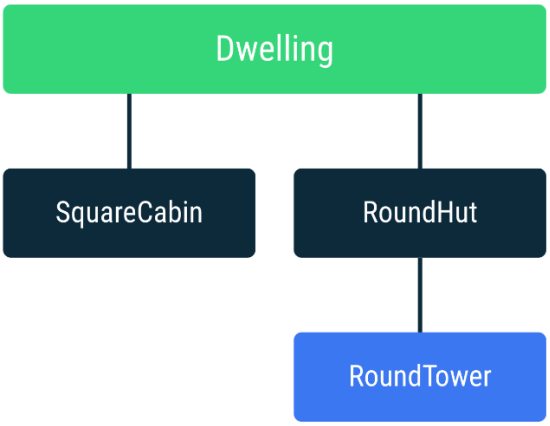
Practical class # 5 – Classes and Inheritance in Kotlin

In this practical class we will discuss how to use classes and inheritance in Kotlin. The class diagram is depicted in this figure.



The classes that will be implementwd:

Dwelling: a base class representing a non-specific shelter that holds information that is common to all dwellings.

SquareCabin: a square cabin made of wood with a square floor area.

RoundHut: a round hut that is made of straw with a circular floor area, and the parent of RoundTower.

RoundTower: a round tower made of stone with a circular floor area and multiple stories.

1) Open the Kotlin playground.

Delete the println function in the main and then declare an abstract class outside the main function. This class should contain two properties, one for the building material and one for the capacity. Moreover, the class accepts as private property the number of residents. Add an hasRoom function to the class to check if there is room for another resident.

The code should look like this.

```
abstract class Dwelling(private var residents: Int) {

  abstract val buildingMaterial: String
  abstract val capacity: Int

  fun hasRoom(): Boolean {
    return residents < capacity
  }
}
```

Try to create an object of type Dwelling to check that this generates and error.

2) Create a squareCabin subclass

Below the Dwelling class, create a squareCabin subclass, which inherits from Dwelling. Remember to pass the parameters expected by the superclass constructor:

```
class squareCabin : Dwelling(3)
```

or evene better

```
class SquaredCabin(residents: Int) : Dwelling(residents)
```

Run the code to see the error raised by the compiler.

3) Define the abstract members inside the subclass

```
class SquaredCabin(residents: Int) : Dwelling(residents){
        override val buildingMaterial = "Wood"
        override val capacity = 6
}
```

Running the code results in no errors.

4) Create an instance of squareCabin in the main function and test its functionalities

```
fun main() {
  val myCabin = SquareCabin(6)
  println("\nSquared Cabin\n============")
  println("Capacity: ${myCabin.capacity}")
  println("Building Material: ${myCabin.buildingMaterial}")
  println("Has room: ${myCabin.hasRoom()}")
}
```

5) Use with to semplify the code

```
fun main() {
        val myCabin = SquareCabin(6)
        with(myCabin){
                println("\nSquared Cabin\n============")
                println("Capacity: ${capacity}")
                println("Building Material: ${buildingMaterial}")
                println("Has room: ${hasRoom()}")
        }
}
```

6) Create a RoundHut subclass

The class should specify “Straw” as building material and 3 as capacity.

7) Create a RoundTower subclass

The building material is “Stone” and the capacity is 4. Run the code to see the error. Declare the superclass as open to solve the error. Add multiple floors to the RoundTower class (optionally you can also assign a default value to this property). Use this value to define a new capacity.

```
class RoundTower(residents: Int, val floors: Int = 2) : RoundHut(residents){
        override val buildingMaterial = "Stone"
        override val capacity = 4*floors
}
```

8) Modify the class hierarchy to include a function for area calculation

Define an abstract function in the abstract class and implement it according to the definition in each subclass.

```
abstract fun floorArea() : Double
```

For the SquaredCabin, insert a new property called length.

```
class SquareCabin(residents: Int, val length: Double) : Dwelling(residents){
    ....
    override fun floorArea(): Double{
        return length * lenght
    }
}
```

For the RoundHut use kotlin.math.PI as constant or import kotlin.math.PI and use PI in the code

9) Write a function to allow a new resident to get a room (if there is space)

In the abstract class, insert this function

```
fun getRoom(){
    if (capacity > residents) {
        residents++
        println("You got a room!")
     } else {
        println("Sorry, at capacity and no rooms left.")
     }
}
```

10) Fit a carpet in the Dwellings

Create a MaxCarpetLenght for each Dwelling in the proper way. If you need to calculate ethe squared root use kotlin.math.sqrt