1) Las clases en Kotlin se declaran usando la keyword class. La declaración incluye el nombre de la clase, seguido del encabezado que incluye los parámetros de tipo y constructor primario; y el cuerpo de la clase donde se declaran los métodos y atributos.

```
class Person { /*...*/ }
```

Las clases pueden tener un constructor primario y uno o más constructores secundarios. Los constructores primarios van en el encabezado después del nombre y los parámetros de tipo opcionales y anotaciones

```
class Person constructor(firstName: String) { /*...*/ }
```

Si el constructor no tiene anotaciones ni modificadores de visibilidad, la keyword constructor puede ser omitida

```
class Person(firstName: String)*...*/ }
```

El constructor primario no contiene código. El código de inicialización se coloca en bloques de inicialización en el cuerpo de la clase, prefijados con la keyword init. Pueden haber varios bloques que se ejectuan en el orden en el que aparecen

```
class InitOrderDemo(name: String) {
    val firstProperty = "First property: $name".also(::println)

    init {
        println("First initializer block that prints $name")
    }

    val secondProperty = "Second property:
${name.length}".also(::println)

    init {
        println("Second initializer block that prints
${name.length}")
    }
}
```

Los parámetros del constructor primario pueden ser utilizados en los bloques de inicialización así como en los inicializadores de atributos. También se puede declarar e inicializar atributos desde el constructor primario, incluyendo valores por defecto

```
class Person(val firstName: String, val lastName: String, var
isEmployed: Boolean = true)
```

Los atributos declarados en el constructor pueden ser mutables (var) o solo de lectura (val).

Una clase también puede tener constructores secundarios que tienen como prefijo la keyword constructor

```
class Person(val pets: MutableList<Pet> = mutableListOf())

class Pet {
    constructor(owner: Person) {
        owner.pets.add(this) // adds this pet to the list of its
owner's pets
    }
}
```

Los constructores secundarios deben delegar al constructor primario, ya sea directamente o indirectamente mediante delegación a otros constructores secundarios. Las delegaciones a otros constructores se hacen con la keyword this

```
class Person(val name: String) {
    val children: MutableList<Person> = mutableListOf()
    constructor(name: String, parent: Person) : this(name) {
        parent.children.add(this)
    }
}
```

Los constructores secundarios ejecutan primero al constructor primario y todos los bloques de inicialización antes de correr sus propios bloques de código

Para crear una instancia de una clase, se llama al constructor como si fuera una función. No es necesario usar la keyword new

```
val invoice = Invoice()

val customer = Customer("Joe Smith")
```

Las clases pueden tener métodos que tiene referencia a todos los miembros de la case sin necesidad de usar la keyword this para accederlos. También puede tener atributos a los que se les pueden definir setters y getters

```
var stringRepresentation: String
    get() = this.toString()
    set(value) {
        setDataFromString(value) // parses the string and assigns
values to other properties
    }
```

Se pueden definir es anidadas

```
class Outer {
    private val bar: Int = 1
    class Nested {
        fun foo() = 2
    }
}

val demo = Outer.Nested().foo() // == 2
```

que pueden tener referencia a la clase de afuera usando keyword inner

```
class Outer {
    private val bar: Int = 1
    inner class Inner {
        fun foo() = bar
    }
}

val demo = Outer().Inner().foo() // == 1
```

Por último también pueden contener objetos anónimos que no necesitan una definición de clase para ser instanciados

```
val helloWorld = object {
   val hello = "Hello"
   val world = "World"
   // object expressions extend Any, so `override` is required on `toString()`
   override fun toString() = "$hello $world"
}
```

2) Puesto que Kotlin es un lenguaje que compila a Java, implementa el recolector de basura de la máquina virtual de Java. Las implementaciones de recolección de basura se basan en dos pasos, que son

Marcado: se identifican los espacios de memoria que están siendo utilizados y los que no

Barrido: se eliminan aquellos objetos identificados durante el paso de marcado

Esto permite el manejo de alojamiento y desalojamiento de de memoria automático

3) La asociación estática ocurre sobre métodos estáticos, finales y privados. La asociación dinámica por su parte ocurre con métodos sobreescritos por class hijas