Objects,
object expressions &
companion objects

`object` = singleton

```
object KSingleton {
    fun foo() {}
}
```

```
KSingleton.foo()
```

object = singleton

```java
public class JSingleton {
    public final static JSingleton INSTANCE = new JSingleton();

    private JSingleton() {}

    public void foo() {}
}
```

```kotlin
object KSingleton {
    fun foo() {}
}
```

Using Kotlin `object` from Java

```java
public class UsingKotlinObjectFromJava {
    public static void main(String[] args) {
        JSingleton.INSTANCE.foo();
        KSingleton.INSTANCE.foo();
    }
}
```

Using Kotlin `object`

```
public class UsingKotlinObjectFromJava {
    public static void main(String[] args) {
        JSingleton.INSTANCE.foo();
        KSingleton.INSTANCE.foo();
    }
}
```

```
KSingleton.foo()
```

object expression

object expressions

replace Java’s anonymous classes

```
window.addMouseListener(
        object : MouseAdapter() {
            override fun mouseClicked(e: MouseEvent) {
                // ...
            }

            override fun mouseEntered(e: MouseEvent) {
                // ...
            }
        }
)
```

Is object expression a singleton?

1. yes
2. no

```
window.addMouseListener(
        object : MouseAdapter() {
            override fun mouseClicked(e: MouseEvent) {
                // ...
            }

            override fun mouseEntered(e: MouseEvent) {
                // ...
            }
        }
)
```

?

Is object expression a singleton?

1. yes
[✓ 2. no]

```
window.addMouseListener(
        object : MouseAdapter() {
            override fun mouseClicked(e: MouseEvent) {
                // ...
            }

            override fun mouseEntered(e: MouseEvent) {
                // ...
            }
        }
)
```

A new instance of object expression is created for each call

```
fun registerTestRepository(customers: Map<Int, Customer>) {
    registerRepository(object : Repository {
        override fun getById(id: Int): Customer? {
            return customers[id]
        }

        override fun getAll(): List<Customer> {
            return customers.values.toList()
        }
    })
}
```

companion object

companion object

special object inside a class

```
class A {
    companion object {
        fun foo() = 1
    }
}

fun main(args: Array<String>) {
    A.foo()
}
```

companion object

can implement an interface

```
interface Factory<T> {
    fun create(): T
}

class A {
    private constructor()

    companion object : Factory<A> {
        override fun create(): A {
            return A()
        }
    }
}
```

`companion object` can be a receiver of extension function

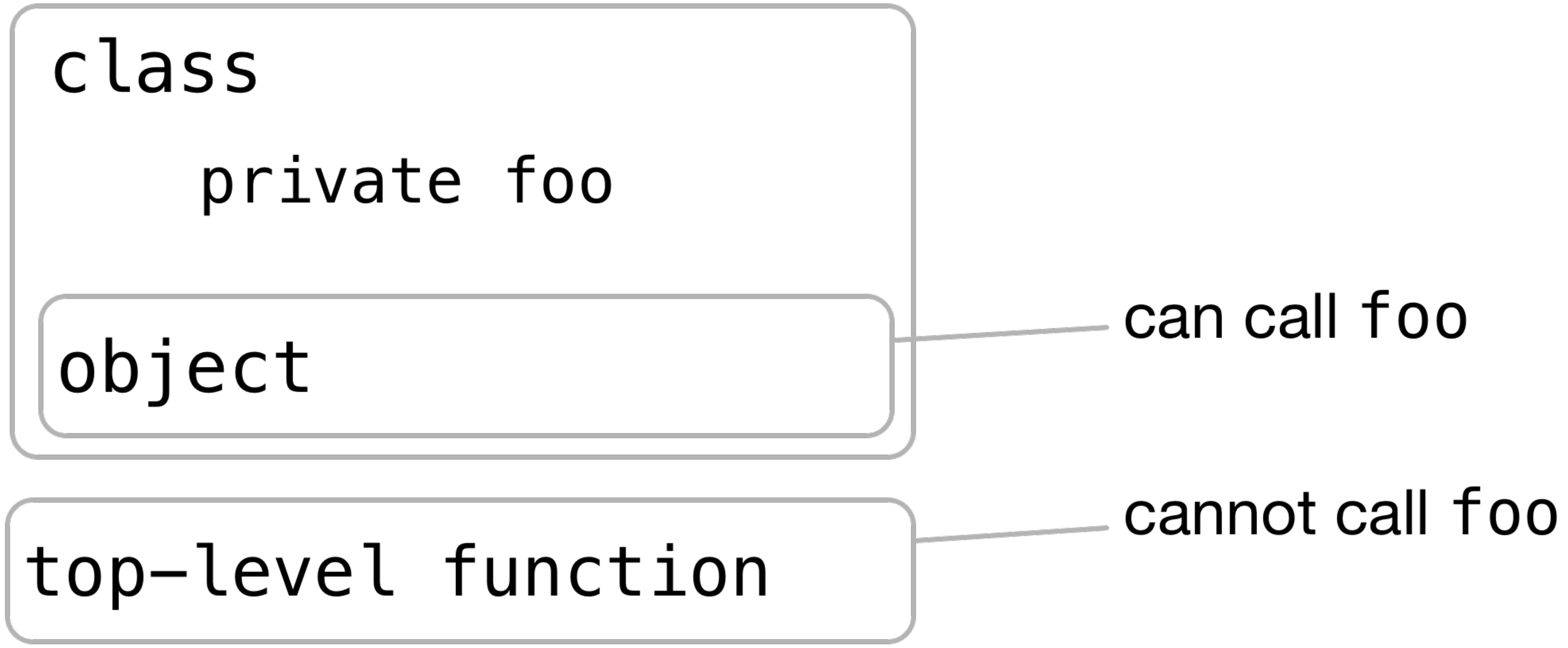
```kotlin
// business logic module
class Person(val firstName: String, val lastName: String) {
    companion object { ... }
}

// client/server communication module
fun Person.Companion.fromJSON(json: String): Person {
    ...
}


val p = Person.fromJSON(json)
```

No `static` keyword

Declare “static” members:

- <u>at the top-level</u>
- inside objects
- inside companion objects

`object` inside a class vs functions outside of the class



@JvmStatic

```
class C {
    companion object {
        @JvmStatic fun foo() {}
        fun bar() {}
    }
}
```

Which line(s) won't compile?

```
class C {
    companion object {
        @JvmStatic fun foo() {}
        fun bar() {}
    }
}

    // Java
#1  C.foo();
#2  C.bar();
#3  C.Companion.foo();
#4  C.Companion.bar();
```

?

Which line(s) won't compile?

```
class C {
    companion object {
        @JvmStatic fun foo() {}
        fun bar() {}
    }
}

    // Java
#1  C.foo();
#2  C.bar();
#3  C.Companion.foo();
#4  C.Companion.bar();
```

#2

@JvmStatic

```
object Obj {
    @JvmStatic fun foo() {}
    fun bar() {}
}
```

```
// Java
Obj.foo();
Obj.bar();
Obj.INSTANCE.foo();
Obj.INSTANCE.bar();
```

Is it possible to declare an inner object?

```
class A {
    inner object B
}
```

1. yes
2. no

?

Is it possible to declare an inner object?

```
class A {
    inner object B
}
```

```
  Compiler error: Modifier 'inner'
    is not applicable to 'object'
```

1. yes
2. [✓ no]

Nested object

```
class A {
    object B
}
```

✓