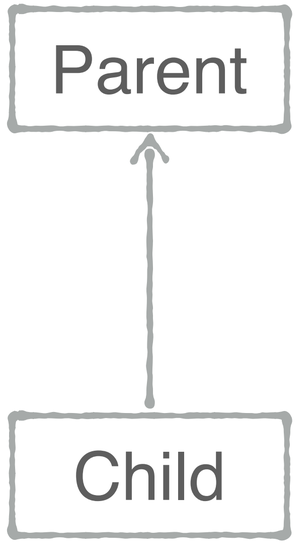
Calling extensions

Extensions to `Child` & `Parent`

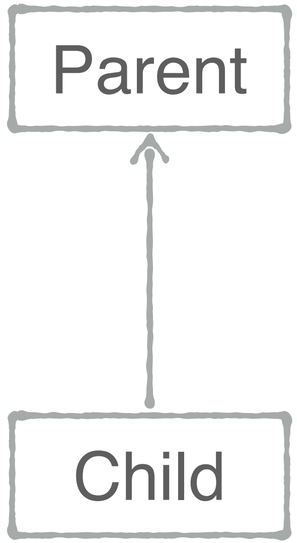
```
open class Parent
class Child: Parent()
```



Extensions to `Child` & `Parent`

```
open class Parent
class Child: Parent()

fun Parent.foo() = "parent"
fun Child.foo() = "child"
```

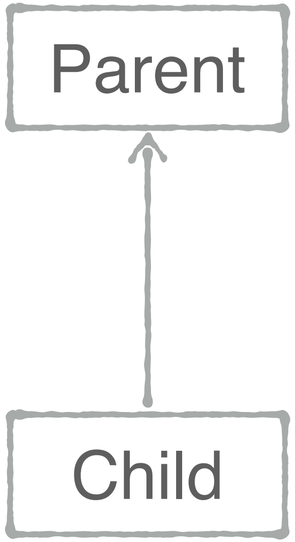


Which function will be called?

```
open class Parent
class Child: Parent()

fun Parent.foo() = "parent"
fun Child.foo() = "child"

fun main(args: Array<String>) {
    val parent: Parent = Child()
    println(parent.foo())
}
```

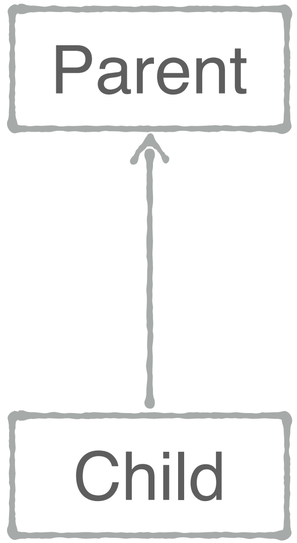


What will be printed?

```
open class Parent
class Child: Parent()

fun Parent.foo() = "parent"
fun Child.foo() = "child"

fun main(args: Array<String>) {
    val parent: Parent = Child()
    println(parent.foo())
}
```



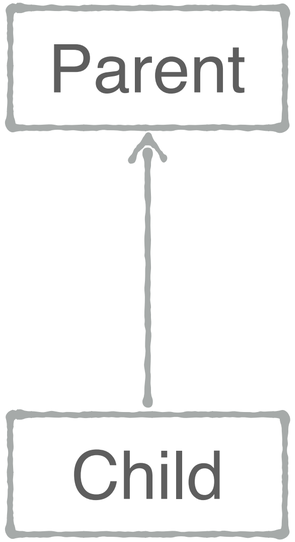
1. parent
2. child

What will be printed?

```
open class Parent
class Child: Parent()

fun Parent.foo() = "parent"
fun Child.foo() = "child"

fun main(args: Array<String>) {
    val parent: Parent = Child()
    println(parent.foo())
}
```



1. parent
2. child

The analogous code in Java

```
public static String foo(Parent parent) { return "parent"; }

public static String foo(Child child) { return "child"; }

public static void main(String[] args) {
    Parent parent = new Child();
    System.out.println(foo(parent));
}
```

parent

The analogous code in Java

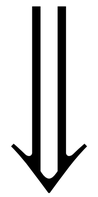
```
public static String foo(Parent parent) { return "parent"; }

public static String foo(Child child) { return "child"; }

public static void main(String[] args) {
    Parent parent = (new Random().nextBoolean())
            ? new Parent() : new Child();
    System.out.println(foo(parent));
}
```

parent

Extensions are `static` Java functions under the hood



No `override` for extension functions in Kotlin

Member vs extension

What will be printed?

```
fun String.get(index: Int) = '*'

fun main(args: Array<String>) {
    println("abc".get(1))
}
```

1. *
2. a
3. b

What will be printed?

```
fun String.get(index: Int) = '*'

fun main(args: Array<String>) {
    println("abc".get(1))
}
```

1. *
2. a
3. b

Extensions don’t hide members

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

fun A.foo() = 2
```

Warning: Extension is shadowed by a member

```
A().foo()   // 1
```

But extensions can overload members

```
class A {
    fun foo() = "member"
}

fun A.foo(i: Int) = "extension($i)"


A().foo(2)    // extension(2)
```