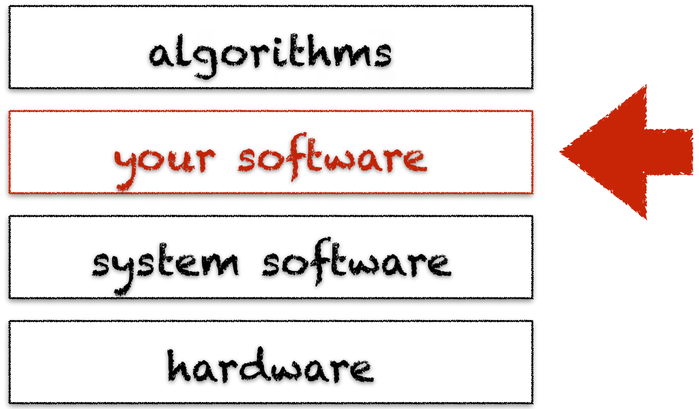
inheritance & polymorphism

learning objectives



- learn how to factor code thanks to inheritance
- learn how to override constructors and methods
- learn about subclassing, subtyping & polymorphism

a vintage example

catalog of books & vinyls

title
author
rating

title
artist
duration
rating

list of books
list of vinyls

a vintage example

catalog of books & vinyls

Book
title author rating
init rate printInfo

Vinyl
title artist duration rating
init rate printInfo

Catalog
books vinyls
addBook addVinyl listInfo

a vintage example

catalog of books & vinyls

catalog of books & vinyls

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.rating = -1

    def rate(self, rating):
        self.rating = rating

    def printInfo(self):
        rating = "?"
        if self.rating >= 0:
            rating = ""
            for i in range(0, self.rating):
                rating = rating + "*"

        print("Book[title: {0} | author: {1} | rating: {2}]"
              .format(self.title, self.author, rating))
```

```
class Vinyl:
    def __init__(self, title, artist, duration):
        self.title = title
        self.artist = artist
        self.duration = duration
        self.rating = -1

    def rate(self, rating):
        self.rating = rating

    def printInfo(self):
        rating = "?"
        if self.rating >= 0:
            rating = ""
            for i in range(0, self.rating):
                rating = rating + "*"

        print("Vinyl[title: {0} | artist: {1} | duration: {2} minutes | rating: {3}]"
              .format(self.title, self.artist, self.duration, rating))
```

catalog of books & vinyls

```
class Catalog:
    def __init__(self):
        self.books = []
        self.vinyls = []

    def addBook(self, book):
        self.books.append(book)

    def addVinyl(self, vinyl):
        self.vinyls.append(vinyl)

    def listInfo(self):
        print("BOOKS")
        for b in self.books:
            b.printInfo()
        print("------")
        print("VINYLS")
        for v in self.vinyls:
            v.printInfo()
```

```
c = Catalog()

b1 = Book("Bible", "God")
b1.rate(3)
b2 = Book("Hamlet", "Shakespeare")

c.addBook(b1)
c.addBook(b2)

v1 = Vinyl("Abbey Road", "Beatles", 47)
v2 = Vinyl("Synchronicity", "Police", 40)

c.addVinyl(v1)
c.addVinyl(v2)

c.listInfo()
```

```
BOOKS
Book[title: Bible | author: God | rating: ***]
Book[title: Hamlet | author: Shakespeare | rating: ?]
------
VINYLS
Vinyl[title: Abbey Road | artist: Beatles | duration: 47 minutes | rating: ?]
Vinyl[title: Synchronicity | artist: Police | duration: 40 minutes | rating: ?]
```

catalog of books & vinyls

```
class Book {
    let title: String
    let author: String
    var rating: Int

    init(title: String, author: String) {
        self.title = title
        self.author = author
        self.rating = -1
    }

    func rate(rating: Int) {
        self.rating = rating
    }

    func printInfo() {
        var rating = "?"

        if (self.rating >= 0) {
            rating = ""
            for i in 1...self.rating {
                rating = rating + "*"
            }
        }
        print("Book[title: \(self.title) | author: \(self.author) | rating: \(rating)]")
    }
}
```

catalog of books & vinyls

```
class Vinyl {
  let title: String
  let artist: String
  let duration: Int
  var rating: Int

  init(title: String, artist: String, duration: Int) {
    self.title = title
    self.artist = artist
    self.duration = duration
    self.rating = -1
  }

  func rate(rating: Int) {
    self.rating = rating
  }

  func printInfo() {
    var rating = "?"

    if (self.rating >= 0) {
      rating = ""
      for i in 1...self.rating {
        rating = rating + "*"
      }
    }
    print("Vinyl[title: \(self.title) | artist: \(self.artist) | duration: \(self.duration) minutes | rating: \(rating)]")
  }
}
```

catalog of books & vinyls

```
class Catalog {
    var books: [Book] = []
    var vinyls: [Vinyl] = []

    func addBook(book:Book) {
        books.append(book)
    }

    func addVinyl(vinyl:Vinyl) {
        vinyls.append(vinyl)
    }

    func listInfo() {
        print("BOOKS")
        for b in books {
            b.printInfo()
        }
        print("------")
        print("VINYLS")
        for v in vinyls {
            v.printInfo()
        }
    }
}
```

```
let c = Catalog()

let b1 = Book(title:"Bible", author:"God")
b1.rate(rating:3)
let b2 = Book(title:"Hamlet", author:"Shakespeare")

c.addBook(book: b1)
c.addBook(book: b2)

let v1 = Vinyl(title:"Abbey Road", artist:"Beatles", duration:47)
let v2 = Vinyl(title:"Synchronicity", artist:"Police", duration:40)

c.addVinyl(vinyl: v1)
c.addVinyl(vinyl: v2)

c.listInfo()
```

```
BOOKS
Book[title: Bible | author: God | rating: ***]
Book[title: Hamlet | author: Shakespeare | rating: ?]
------
VINYLS
Vinyl[title: Abbey Road | artist: Beatles | duration: 47 minutes | rating: ?]
Vinyl[title: Synchronicity | artist: Police | duration: 40 minutes | rating: ?]
```

what's wrong?

```swift
class Vinyl {
  let title: String
  let artist: String
  let duration: Int
  var rating: Int

  init(title: String, artist: String, duration: Int) {
    self.title = title
    self.artist = artist
    self.duration = duration
    self.rating = -1
  }

  func rate(rating: Int) {
    self.rating = rating
  }

  func printInfo() {
    var rating = "?"

    if (self.rating >= 0) {
      rating = ""
      for i in 1...self.rating {
        rating = rating + "*"
      }
    }
    print("Vinyl[title: \(self.title) | artist: \(self.artist) |
  }
}
```

```swift
class Book {
    let title: String
    let author: String
    var rating: Int

    init(title: String, author: String) {
        self.title = title
        self.author = author
        self.rating = -1
    }

    func rate(rating: Int) {
        self.rating = rating
    }

    func printInfo() {
        var rating = "?"

        if (self.rating >= 0) {
            rating = ""
            for i in 1...self.rating {
                rating = rating + "*"
            }
        }
        print("Book[title: \(self.title) | author: \(self.author) | ra
    }
}
```

the Vinyl and Book classes are very similar → code duplication

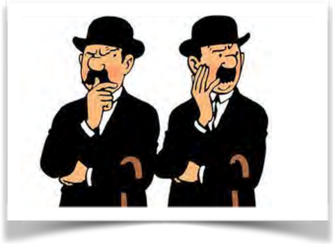
code duplication

...also in the Catalog class

```
class Catalog {
    var books: [Book] = []
    var vinyls: [Vinyl] = []

    func addBook(book:Book) {
        books.append(book)
    }

    func addVinyl(vinyl:Vinyl) {
        vinyls.append(vinyl)
    }

    func listInfo() {
        print("BOOKS")
        for b in books {
            b.printInfo()
        }
        print("------")
        print("VINYLS")
        for v in vinyls {
            v.printInfo()
        }
    }
}
```

maintenance is made difficult

great danger of introducing bugs when evolving



great danger of introducing bugs when evolving

assume we want to support games in v2.0

create a Game class

with many similarities to the Book and Vinyl classes

add a `games` field in Catalog

initialize it in the constructor in Catalog

add an `addGame()` method in Catalog

add a loop in the `listInfo()` method of Catalog

solution: inheritance

inheritance

allows us to define one class, the subclass, as an extension of another, the superclass

inheritance

superclass

defines common attributes

subclasses

defines specific attributes

inherits all fields and methods from its superclass

inheritance

common attributes

Book
title author rating
init rate printInfo

Vinyl
title artist duration rating
init rate printInfo

inheritance

Book

```
title
author
rating

init
rate
printInfo
```

Medium

create a superclass

Vinyl

```
title
artist
duration
rating

init
rate
printInfo
```

inheritance

Medium

```
title
rating
```

```
init
rate
printInfo
```

Book

```
author
```

```
init
```

promote common attributes to the superclass

Vinyl

```
artist
duration
```

```
init
```

subclasses inherit common attributes from the superclass and define their specific attributes

inheritance

```
class Medium:
    def __init__(self, title):
        self.title = title
        self.rating = -1

    def rate(self, rating):
        self.rating = rating

    def printInfo(self):
        rating = "?"
        if self.rating >= 0:
            rating = ""
            for i in range(0, self.rating):
                rating = rating + "*"

        print("Medium[title: {0} | rating: {1}]".format(self.title, rating))
```

```
class Book(Medium):
    def __init__(self, title, author):
        self.author = author
        super().__init__(title)
```

```
class Vinyl(Medium):
    def __init__(self, title, artist, duration):
        self.artist = artist
        self.duration = duration
        super().__init__(title)
```

```
class Catalog:
    def __init__(self):
        self.media = []

    def addMedium(self, medium):
        self.media.append(medium)

    def listInfo(self):
        print("MEDIA")
        for m in self.media:
            m.printInfo()
```

```
c = Catalog()
b = Book("Hamlet", "Shakespeare")
v = Vinyl("Abbey Road", "Beatles", 47)
b.rate(3)

c.addMedium(b)
c.addMedium(v)

c.listInfo()
```

inheritance

```
class Medium {
    let title: String
    var rating: Int

    init(title: String) {
        self.title = title
        self.rating = -1
    }

    func rate(rating: Int) {
        self.rating = rating
    }

    func printInfo() {
        var rating = "?"

        if (self.rating >= 0) {
            rating = ""
            for i in 1...self.rating {
                rating = rating + "*"
            }
        }
        print("Medium[title: \(self.title) | rating: \(rating)]")
    }
}
```

```
class Book : Medium {
    let author: String

    init(title: String, author: String) {
        self.author = author
        super.init(title: title)
    }
}
```

```
class Vinyl : Medium {
    let artist: String
    let duration: Int

    init(title: String, artist: String, duration: Int) {
        self.artist = artist
        self.duration = duration
        super.init(title: title)
    }
}
```

```
class Catalog {
    var media: [Medium] = []

    func addMedium(medium:Medium) {
        media.append(medium)
    }

    func listInfo() {
        print("MEDIA")
        for m in media {
            m.printInfo()
        }
    }
}
```

```
let c = Catalog()
let b = Book(title:"Bible", author:"God")
let v = Vinyl(title:"Abbey Road", artist:"Beatles", duration:47)
b.rate(rating:3)

c.addMedium(medium: b)
c.addMedium(medium: v)

c.listInfo()
```

calling the superclass constructor

this is necessary to initialize the fields inherited from the superclass

subtyping

before

```
class Catalog {
    ...
    func addBook(book:Book) {...}
    func addVinyl(vinyl:Vinyl) {...}
    ...
}
```

after

```
class Catalog {
    ...
    func addMedium(medium:Medium) {...}
  ...
}
```

```
let c = Catalog()
let b = Book(title:"Bible", author:"God")
let v = Vinyl(title:"Abbey Road", artist:"Beatles", duration:47)

c.addMedium(medium: b)
c.addMedium(medium: v)
```

classes define types

a geek

a person with a devotion to something in a way that places him or her outside the mainstream*

*wikipedia

subclasses define subtypes

substitution principle

objects of subtypes can be used where objects of supertypes are required


LINUX GEEK


CODE GEEK


KISS GEEK

polymorphic variables

object variables are **polymorphic** because they can hold objects of **more than one type**

they can hold objects **of the declared type** (`Medium`), **or of subtypes** of the declared type (`Book`, `Vinyl`)

type casting

```
var b = Book(title:"Hamlet", author:"Shakespeare")
var v = Vinyl(title:"Abbey Road", artist:"Beatles", duration:47)
var m : Medium
```

✘ `v = b`

✘ `b = v`

✔ `m = v`

✔ `m = b`

✘ `b = m`

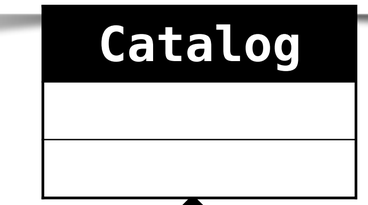
`b = m as! Book` ✔

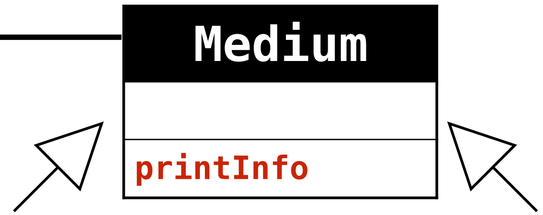
the substitution principle only works...

back to printing books and vinyls

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```

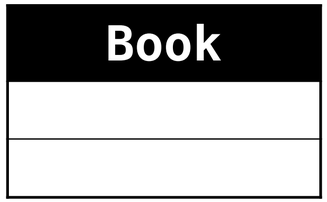
```
func printInfo() {
  var rating = "?"

  if (self.rating >= 0) {
    rating = ""
    for i in 1...self.rating {
      rating = rating + "*"
    }
  }
  print("Medium[title: \(self.title) | rating: \(rating)]")
}
```

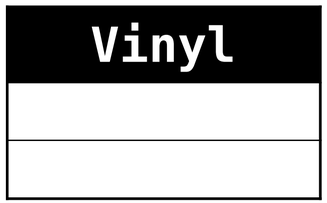





```
Medium[title: Bible | rating: ***]
Medium[title: Synchronicity | rating: ?]
```

the `printInfo` method in Medium only prints the common fields





the problem

inheritance is a one-way street: a subclass inherits the superclass fields

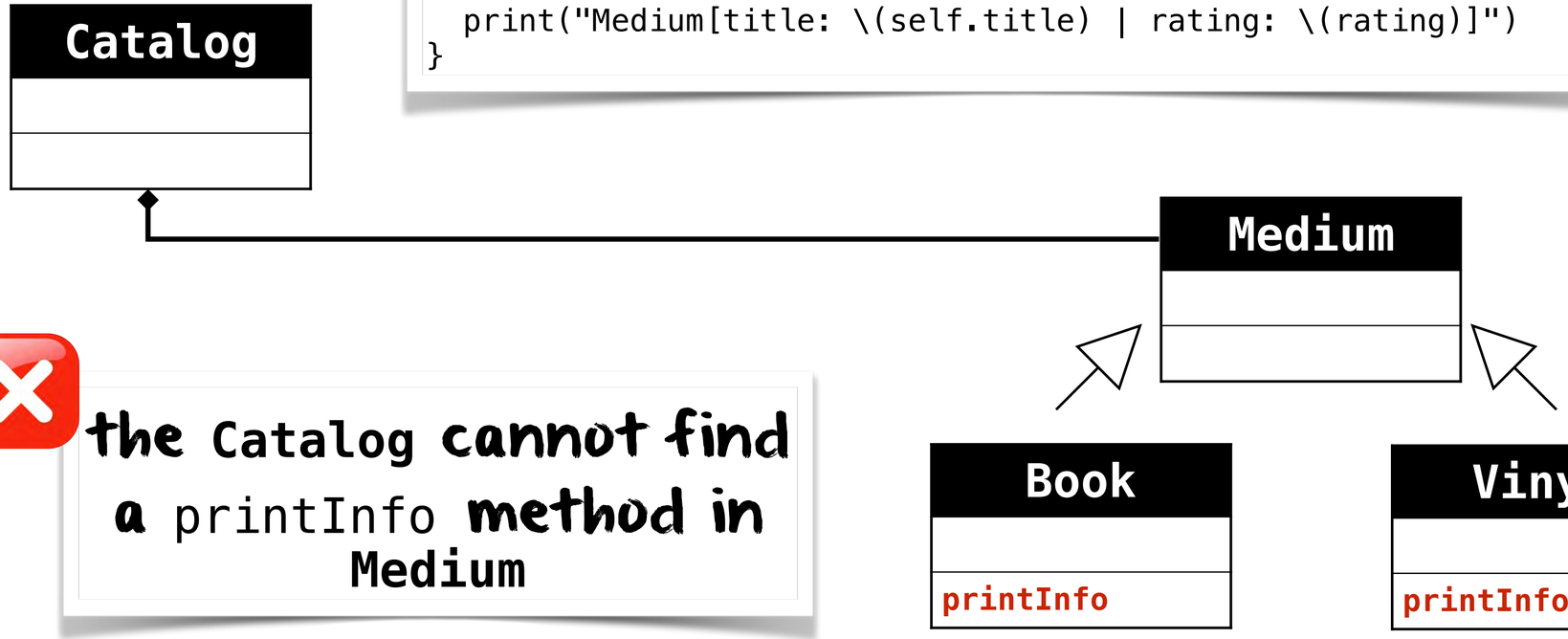
the superclass knows nothing about the fields of its subclasses


try to
move printInfo
down to Book
and Vinyl

printing books & vinyls

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```

```
func printInfo() {
  var rating = "?"

  if (self.rating >= 0) {
    rating = ""
    for i in 1...self.rating {
      rating = rating + "*"
    }
  }
  print("Medium[title: \(self.title) | rating: \(rating)]")
}
```



the Catalog cannot find a printInfo method in Medium

back to polymorphism

static type vs **dynamic type**

what's the type of variable `b` ?

```
var b : Book = Book(title:"Hamlet", author:"Shakespeare")
```

what's the type of variable `m` ?

```
var m : Medium = Book(title:"Hamlet", author:"Shakespeare")
```

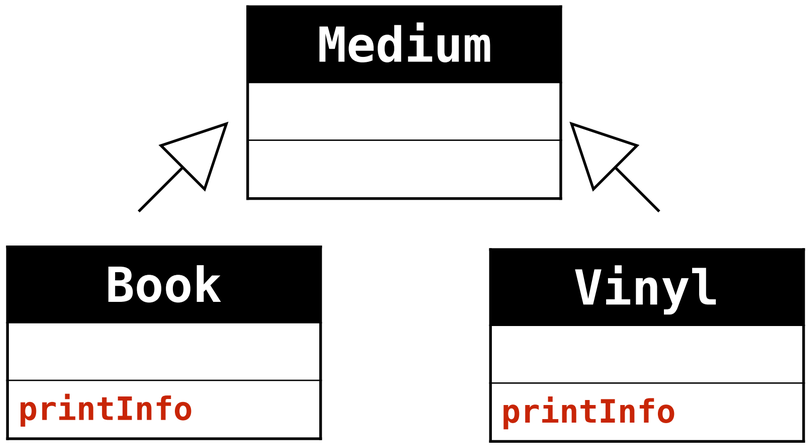
static type

dynamic type

back to polymorphism

what's the type of variable m ?

```
var m : Medium = Book(title:"Hamlet", author:"Shakespeare")
```




```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```

compile-time error!

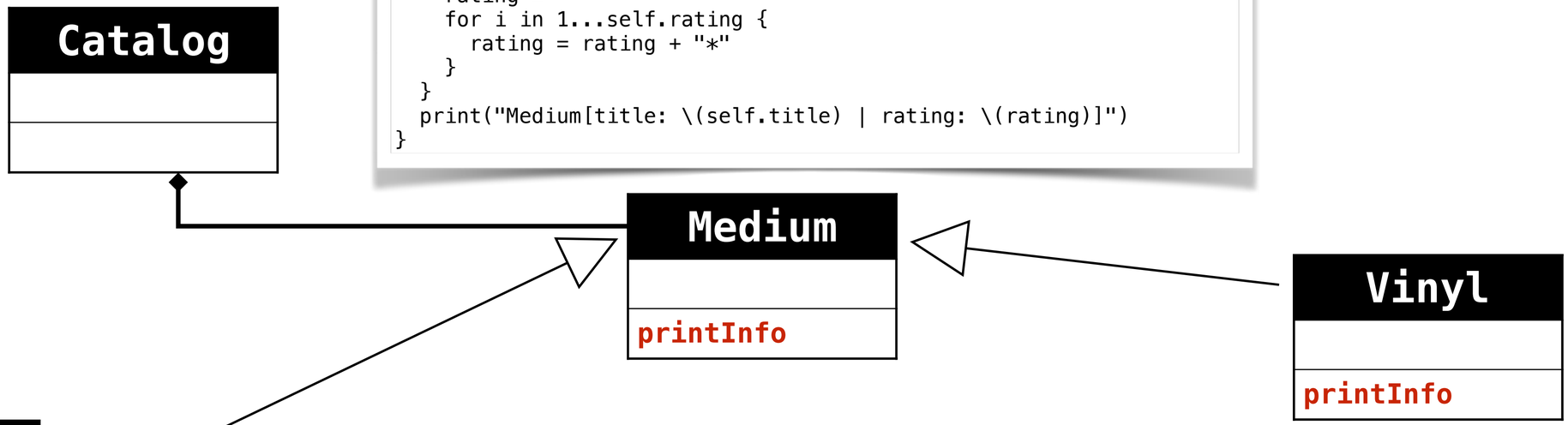
back to work


try to
add printInfo to
Medium, Book and
Vinyl

printing books & vinyls

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```

```
func printInfo() {
  var rating = "?"

  if (self.rating >= 0) {
    rating = ""
    for i in 1...self.rating {
      rating = rating + "*"
    }
  }
  print("Medium[title: \(self.title) | rating: \(rating)]")
}
```

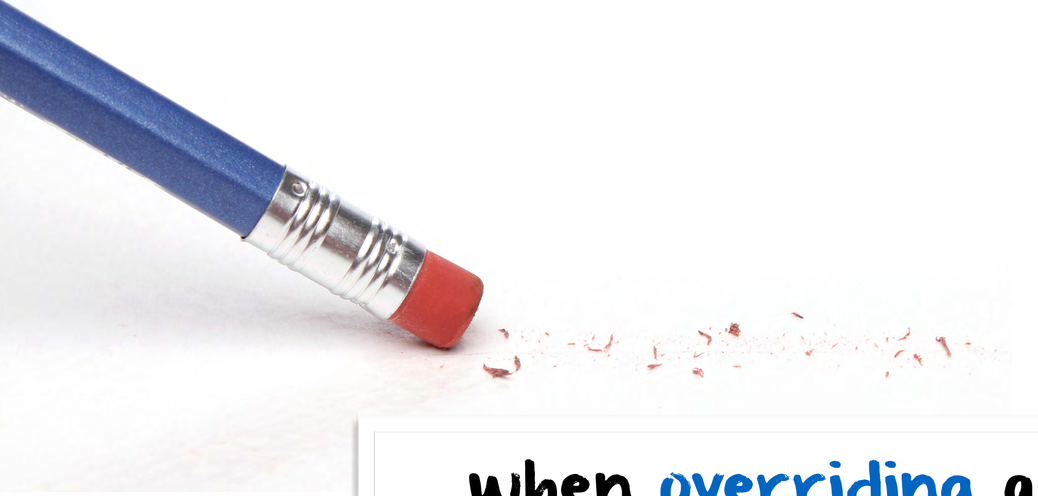



```
override func printInfo() {
  print("Vinyl[artist: \(self.artist) | Duration: \(self.duration)]")
}
```

```
Book[author: Shakespeare]
Vinyl[artist: Beatles | Duration: 47]
```

```
override func printInfo() {
  print("Book[author: \(self.author)]")
}
```

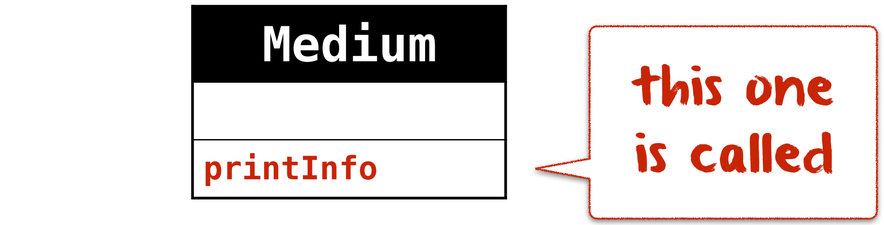
no longer printing common fields

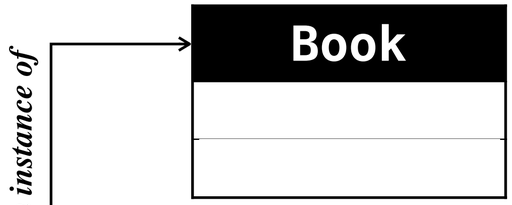


the problem

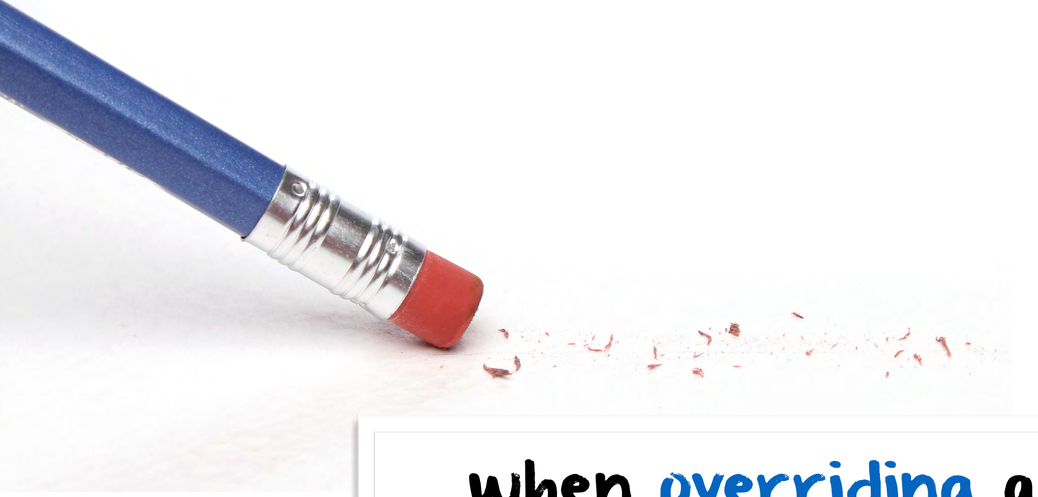
when overriding a method in a subclass (Book or Vinyl), the inherited method from the super class is hidden

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```



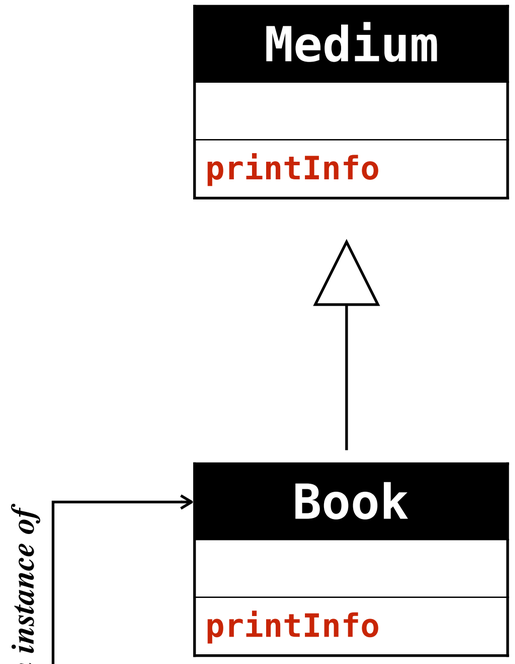


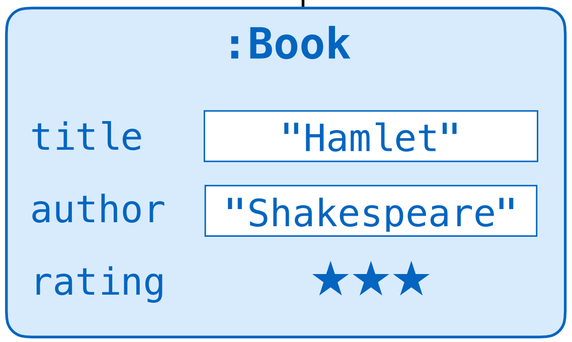
the problem



when overriding a method in a subclass (Book or Vinyl), the inherited method from the super class is hidden

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```





almost there

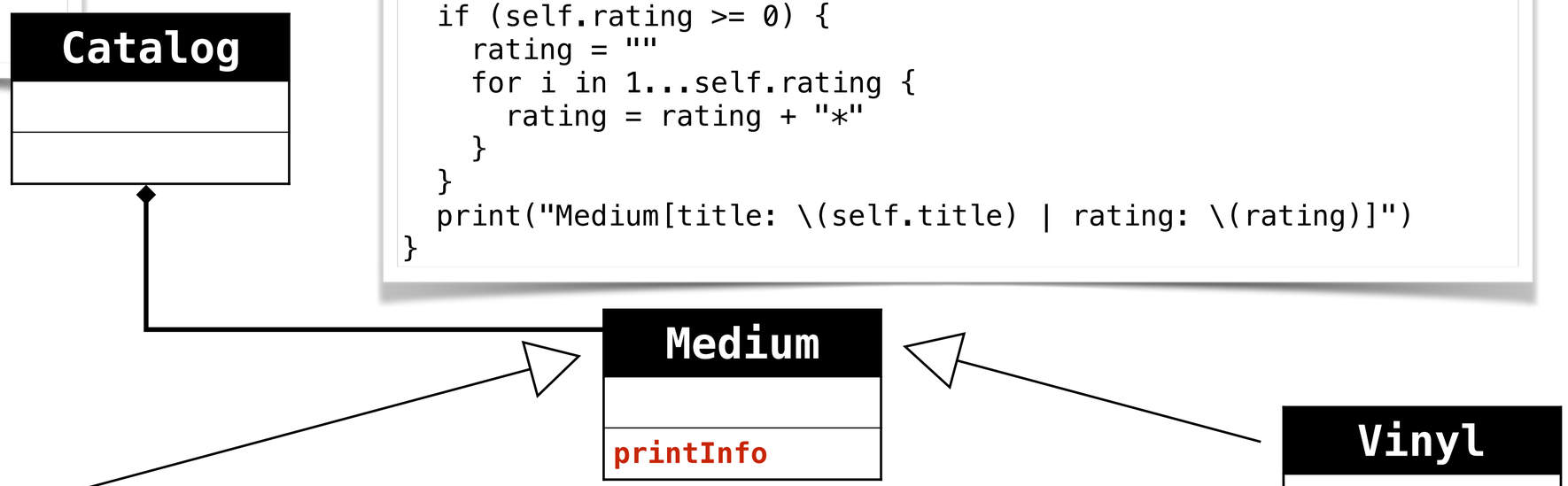
solution

call super

printing books & vinyls

```
func listInfo() {
  print("MEDIA")
  for m in media {
    m.printInfo()
  }
}
```

```
func printInfo() {
  var rating = "?"

  if (self.rating >= 0) {
    rating = ""
    for i in 1...self.rating {
      rating = rating + "*"
    }
  }
  print("Medium[title: \(self.title) | rating: \(rating)]")
}
```




```
override func printInfo() {
  super.printInfo()
  print("Vinyl[artist: \(self.artist) | Duration: \(self.duration)]")
}
```

```
override func printInfo() {
  super.printInfo()
  print("Book[author: \(self.author)]")
}
```

```
Medium[title: Hamlet | rating: ?]
Book[author: Shakespeare]

Medium[title: Abbey Road | rating: ?]
Vinyl[artist: Beatles | Duration: 47]
```

the object superclass

in most object-oriented languages, all classes have an implicit superclass, the `Object` class

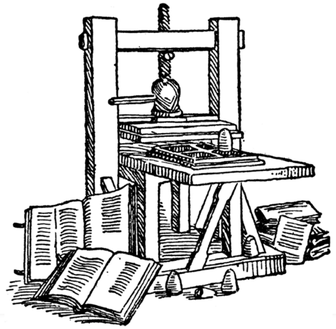
so methods in `Object` are inherited by all classes and can be overridden by any subclasses

for example in Python and in Scala, all objects, inherit from a method to represent themselves as strings and can override it

```python
class Medium:
  def __str__(self):
    return "I am a Medium"
```

```scala
class Medium {
  override def toString: String = {"I am a Medium"}
}
```

back to printing books and vinyls



one last time

I promise

```
class Medium:
    ...
    def __str__(self):
        rating = "?"
        if self.rating >= 0:
            rating = ""
            for i in range(0, self.rating):
                rating = rating + "*"

        return "title: {0} | rating: {1}".format(self.title,
rating)

    def printInfo(self):
        print(str(self))
```

```
class Book(Medium):
    ...
    def __str__(self):
        return "Book[author: {0} | {1}]".format(self.author,
super().__str__())
```

```
class Vinyl(Medium):
  ...
  def __str__(self):
    return "Vinyl[artist: {0} | duration: {1} minutes | {2}]".format(self.artist,
self.duration, super().__str__())
```

```
Book[author: God | title: Bible | rating: ***]
Book[author: Shakespeare | title: Hamlet | rating: ?]
Vinyl[artist: Beatles | duration: 47 minutes | title: Abbey Road | rating: ?]
Vinyl[artist: Police | duration: 40 minutes | title: Synchronicity | rating: ?]
```