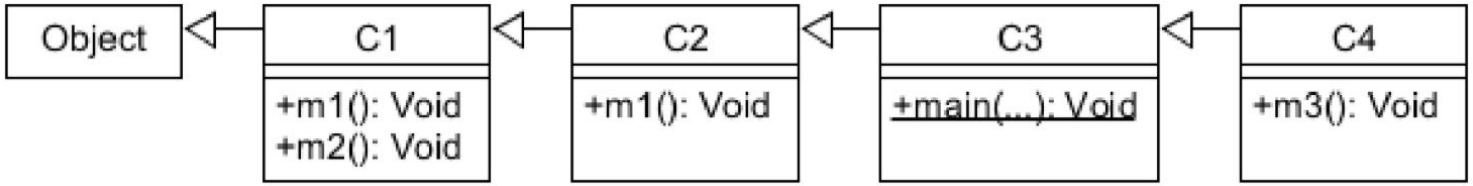
Exercise 5

栗林健太郎 2030006（2020年10月3日）

p.4の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

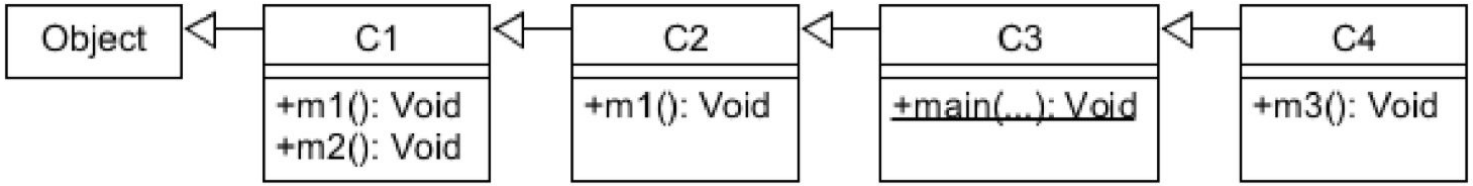
```
(new C4()).m1();
```

 # 1. コンパイル時

- (new C4())の型: C4
- 引数の型 (): 引数なし
- 探索範囲: C4およびそのすべての親クラス
- 候補: m1()
- 付与されるメソッドシグネチャ: m1()

 # 2. 実行時

- (new C4())の型: C4（探索の開始地点）
- 実行されるメソッド: C2のm1()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

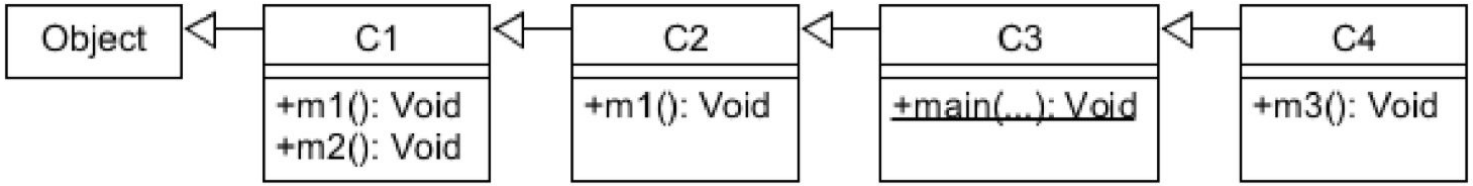
```
(new C4()).m2();
```

 # 1. コンパイル時

- (new C4())の型: C4
- 引数の型 (): 引数なし
- 探索範囲: C4およびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- (new C4())の型: C4（探索の開始地点）
- 実行されるメソッド: C1のm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

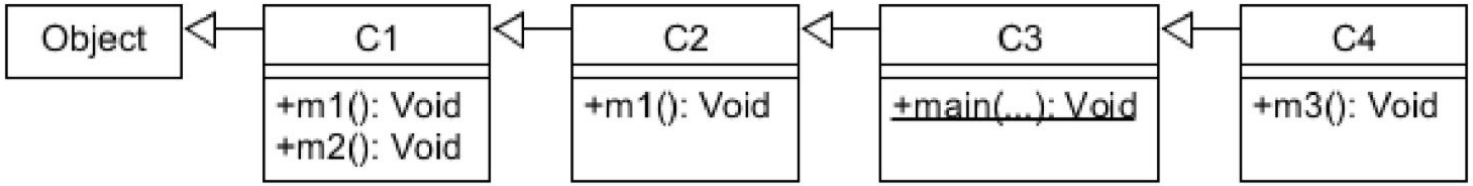
```
(new C4()).m3();
```

 # 1. コンパイル時

- (new C4())の型: C4
- 引数の型 (): 引数なし
- 探索範囲: C4およびそのすべての親クラス
- 候補: m3()
- 付与されるメソッドシグネチャ: m3()

 # 2. 実行時

- (new C4())の型: C4（探索の開始地点）
- 実行されるメソッド: C4のm3()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

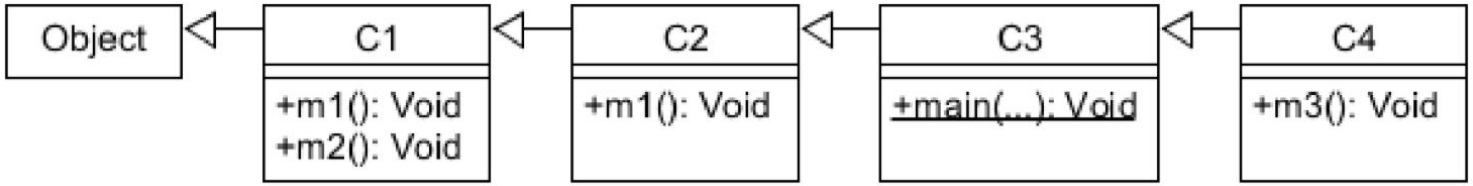
```
C1 o = new C4(); o.m1();
```

 # 1. コンパイル時

- oの型: C1
- 引数の型 (): 引数なし
- 探索範囲: C1およびそのすべての親クラス
- 候補: m1()
- 付与されるメソッドシグネチャ: m1()

 # 2. 実行時

- oの型: C4（探索の開始地点）
- 実行されるメソッド: C2のm1()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

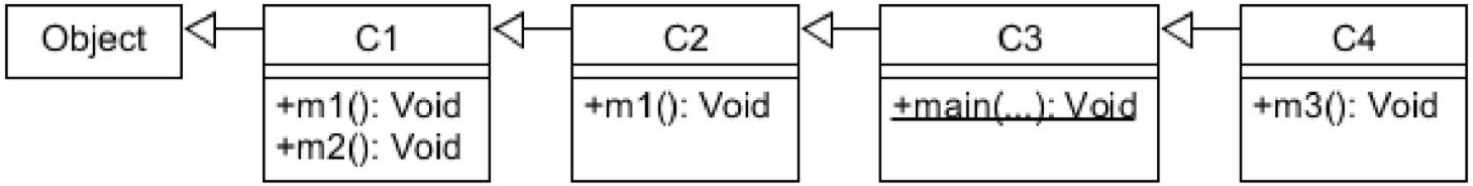
```
C1 o = new C4(); o.m2();
```

 # 1. コンパイル時

- oの型: C1
- 引数の型 (): 引数なし
- 探索範囲: C1およびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- oの型: C4（探索の開始地点）
- 実行されるメソッド: C1のm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
C1 o = new C4(); o.m3();
```

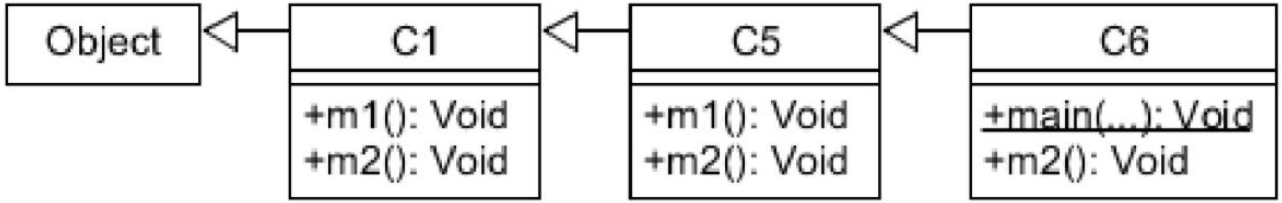
 # 1. コンパイル時

- oの型: C1
- 引数の型 (): 引数なし
- 探索範囲: C1およびそのすべての親クラス
- 候補: なし
- 付与されるメソッドシグネチャ: なし（候補がないため）。
  したがって、コンパイルエラーとなる。

 # 2. 実行時

- コンパイルエラーとなるため実行不可能である。

p.6の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

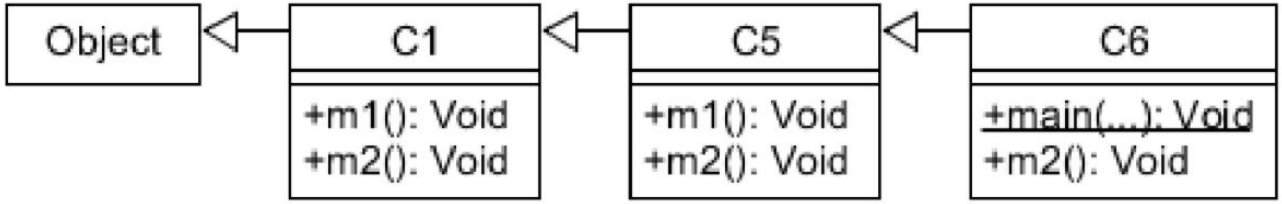
```
(new C6()).m1();
```

 # 1. コンパイル時

- (new C6())の型: C6
- 引数の型 (): 引数なし
- 探索範囲: C6およびそのすべての親クラス
- 候補: m1()
- 付与されるメソッドシグネチャ: m1()

 # 2. 実行時

- (new C6())の型: C6（探索の開始地点）
- 実行されるメソッド: C5のm1()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C6()).m1();
```

 # 1. コンパイル時

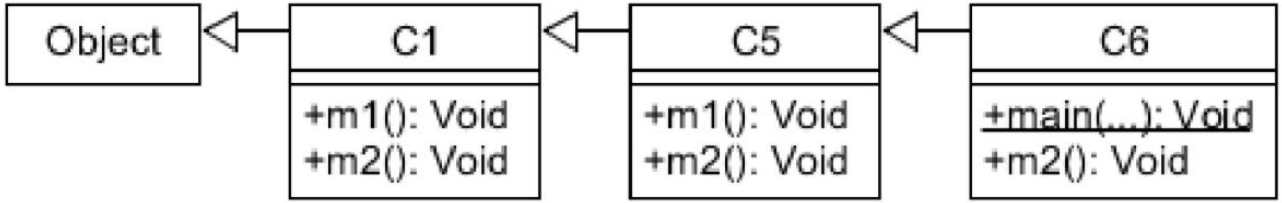
- thisの型: C5
- 引数の型 (): 引数なし
- 探索範囲: C5およびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: C6（探索の開始地点）
- 実行されるメソッド: C6のm2()

```
// C5のm1()が以下の実装だった場合:

public void m1() {
    System.out.println("m1() in C5 was
invoked");
    this.m2();
    super.m2();
}
```



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C6()).m1();
```

 # 1. コンパイル時

- superの型: C1
- 引数の型 (): 引数なし
- 探索範囲: C1およびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

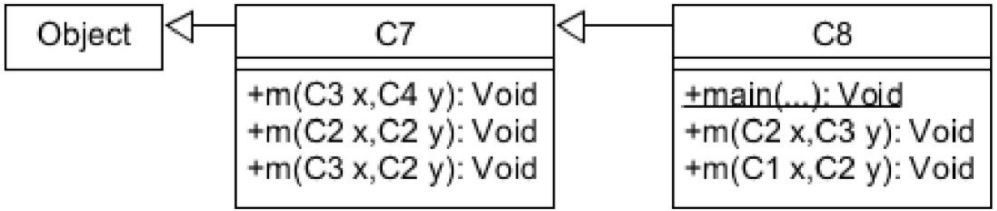
 # 2. 実行時

- superの型: C1（探索の開始地点）
- 実行されるメソッド: C1のm2()

```
// C5のm1()が以下の実装だった場合:

public void m1() {
    System.out.println("m1() in C5 was
invoked");
    this.m2();
    super.m2();
}
```

p.11の例

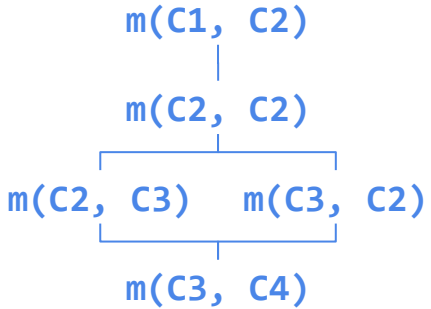


上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C8()).m(new C4(), new C4());
```

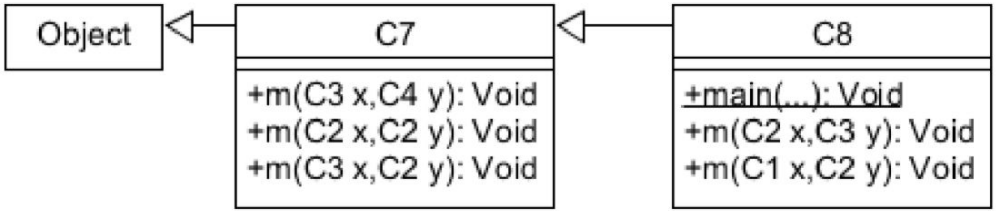
 # 1. コンパイル時

- (new C8())の型: C8
- 引数の型 (new C4(), new C4()): (C4, C4)
- 探索範囲: C8およびそのすべての親クラス
- 候補: m(C1, C2), m(C2,C2), m(C2, C3), m(C3, C2), m(C3, C4)
- 付与されるメソッドシグネチャ: m(C3, C4)（最小要素であるため）



 # 2. 実行時

- (new C8())の型: C8（探索の開始地点）
- 実行されるメソッド: C7のm(C3, C4)



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
C2 x = new C4(); C2 y = new C4(); (new C8()).m(x, y)
```

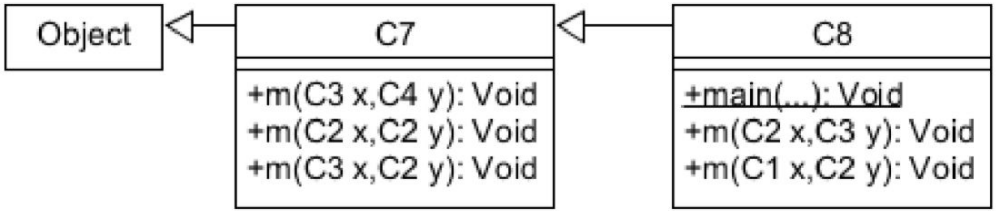
 # 1. コンパイル時

```
m(C1, C2)
    |
m(C2, C2)
```

- (new C8())の型: C8
- 引数の型 (x, y): (C2, C2)
- 探索範囲: C8およびそのすべての親クラス
- 候補: m(C1, C2), m(C2,C2)
- 付与されるメソッドシグネチャ: m(C2, C2)（最小要素であるため）

 # 2. 実行時

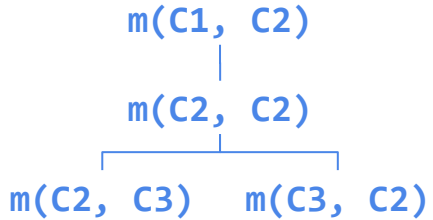
- (new C8())の型: C8（探索の開始地点）
- 実行されるメソッド: C7のm(C2, C2)



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C8()).m(new C3(), new C3());
```

 # 1. コンパイル時

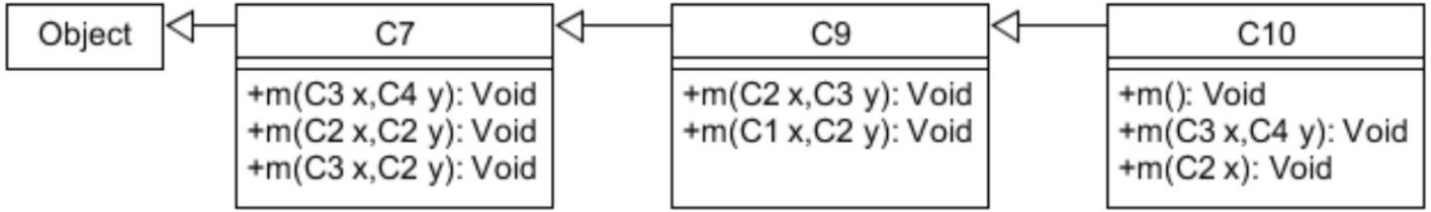


- (new C8())の型: C8
- 引数の型 (new C3(), new C3()): (C3, C3)
- 探索範囲: C8およびそのすべての親クラス
- 候補: m(C1, C2), m(C2,C2), m(C2, C3), m(C3, C2), m(C3, C4)
- 付与されるメソッドシグネチャ: なし(最小要素がないため)。
  したがって、コンパイルエラーとなる。

 # 2. 実行時

- コンパイルエラーとなるため実行不可能である。

p.12の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

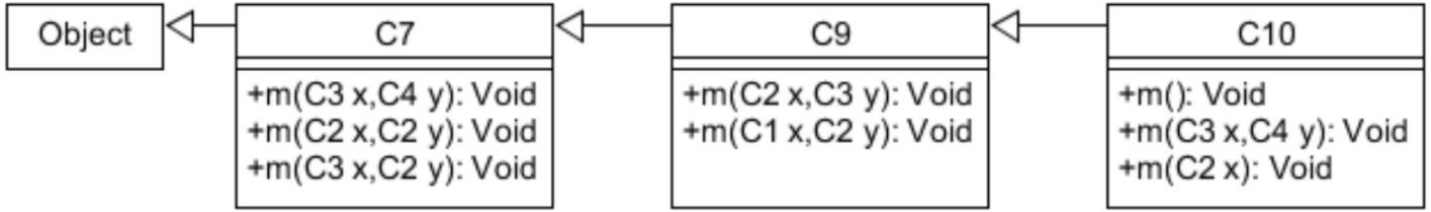
```
(new C10()).m()
```

 # 1. コンパイル時

- (new C10())の型: C10
- 引数の型 (): なし
- 探索範囲: C10およびそのすべての親クラス
- 候補: m()
- 付与されるメソッドシグネチャ: m()

 # 2. 実行時

- (new C10())の型: C10（探索の開始地点）
- 実行されるメソッド: m()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

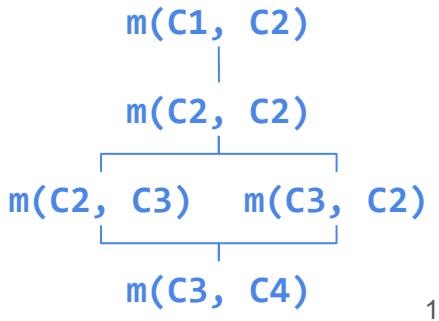
```
(new C10()).m()
```

 # 1. コンパイル時

- thisの型: C10
- 引数の型 (new C4(), new C4()):(C4, C4)
- 探索範囲: C10およびそのすべての親クラス
- 候補: m(C1, C2), m(C2, C2), m(C2, C3), m(C3, C2), m(C3, C4)
- 付与されるメソッドシグネチャ: m(C3, C4)
  (最小要素であるため)

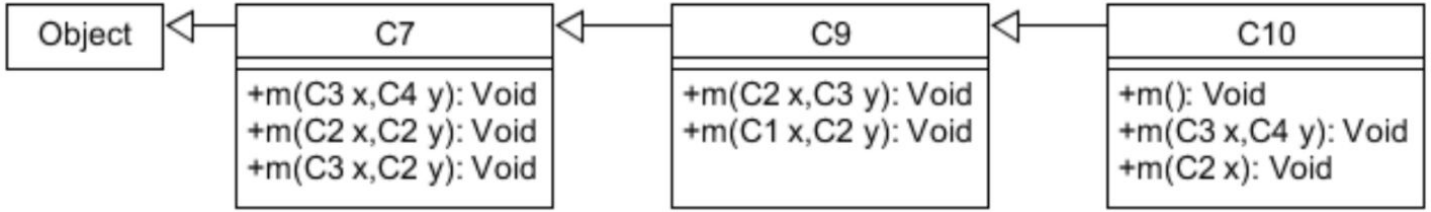
```
// C10のm()が以下の実装だった場合:

public void m1() {
    System.out.println("m() in C10 was invoked");
    this.m(new C4(), new C4());
    super.m(new C4(), new C4());
    this.m(new C3());
    // super.m(new C3());
}
```



 # 2. 実行時

- thisの型: C10(探索の開始地点)
- 実行されるメソッド: C10のm(C3, C4)



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

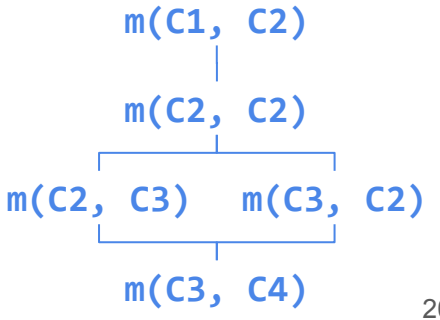
```
(new C10()).m()
```

 # 1. コンパイル時

- superの型: C9
- 引数の型 (new C4(), new C4()):(C4, C4)
- 探索範囲: C9およびそのすべての親クラス
- 候補: m(C1, C2), m(C2, C2), m(C2, C3), m(C3, C2), m(C3, C4)
- 付与されるメソッドシグネチャ: m(C4, C3)
  (最小要素であるため)

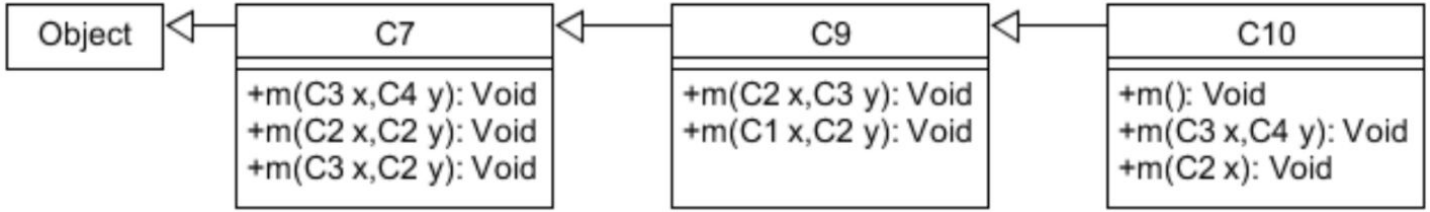
```
// C10のm()が以下の実装だった場合:

public void m1() {
    System.out.println("m() in C10 was invoked");
    this.m(new C4(), new C4());
    super.m(new C4(), new C4());
    this.m(new C3());
    // super.m(new C3());
}
```



 # 2. 実行時

- superの型: C9(探索の開始地点)
- 実行されるメソッド: C7のm(C3, C4)



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C10()).m()
```

 # 1. コンパイル時

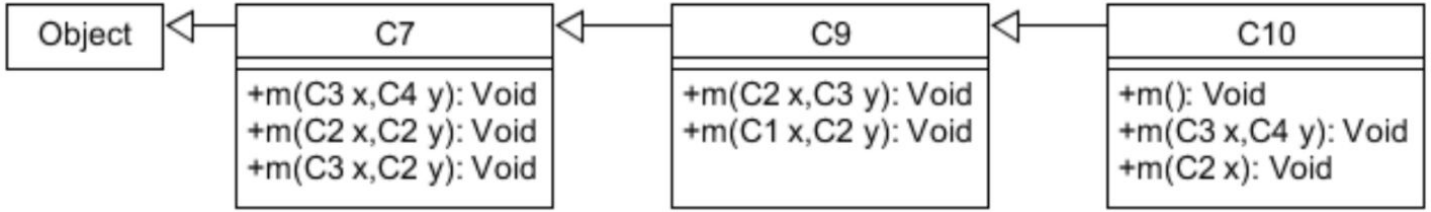
- thisの型: C10
- 引数の型 (new C3()):(C3)
- 探索範囲: C10およびそのすべての親クラス
- 候補: m(C2)
- 付与されるメソッドシグネチャ: m(C2)
  (最小要素であるため)

```
// C10のm()が以下の実装だった場合:

public void m1() {
    System.out.println("m() in C10 was invoked");
    this.m(new C4(), new C4());
    super.m(new C4(), new C4());
    this.m(new C3());
    // super.m(new C3());
}
```

 # 2. 実行時

- thisの型: C10(探索の開始地点)
- 実行されるメソッド: C10のm(C2)



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
(new C10()).m()
```

 # 1. コンパイル時

- superの型: C9
- 引数の型（new C3()):（C3）
- 探索範囲: C9およびそのすべての親クラス
- 候補: なし
- 付与されるメソッドシグネチャ: なし
  （候補がないため）
  したがって、コンパイルエラーとなる。

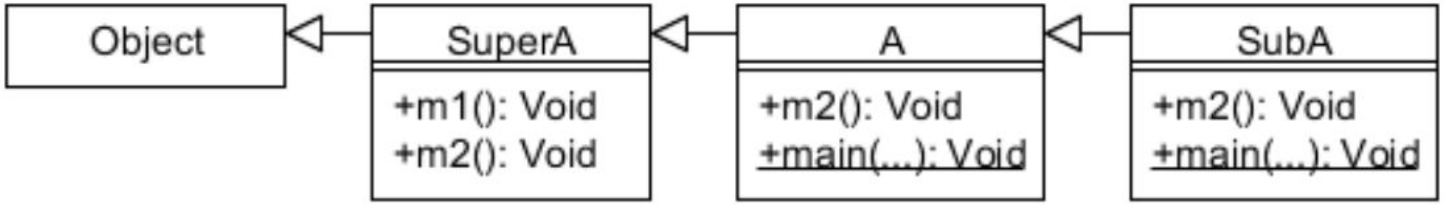
```
// C10のm()が以下の実装だった場合:

public void m1() {
    System.out.println("m() in C10 was invoked");
    this.m(new C4(), new C4());
    super.m(new C4(), new C4());
    this.m(new C3());
    super.m(new C3());  // ←のコメントをはずした
}
```

 # 2. 実行時

- コンパイルエラーとなるため実行不可能である。

p.13の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public void m1() { System.out.print("m1() in SuperA;"); this.m2(); }
public void m2() { System.out.print("m2() in SuperA;"); }
```

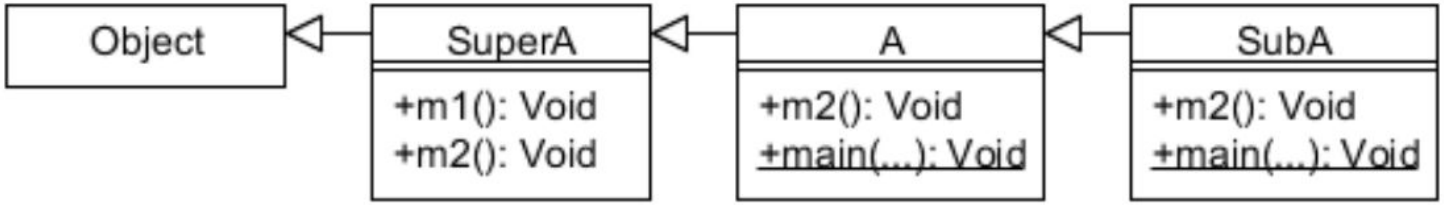
このthisがSubAの場合

 # 1. コンパイル時

- thisの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: SubA（探索の開始地点）
- 実行されるメソッド: SubAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public void m1() { System.out.print("m1() in SuperA;"); this.m2(); }
public void m2() { System.out.print("m2() in SuperA;"); }
```

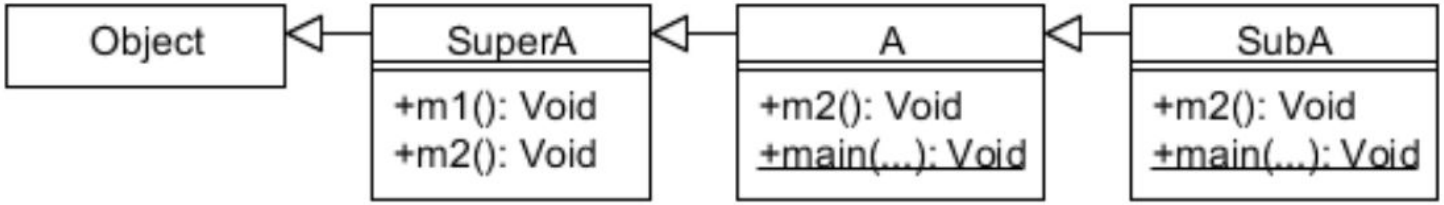
このthisがAの場合

 # 1. コンパイル時

- thisの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: A（探索の開始地点）
- 実行されるメソッド: Aのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public void m1() { System.out.print("m1() in SuperA;"); this.m2(); }
public void m2() { System.out.print("m2() in SuperA;"); }
```

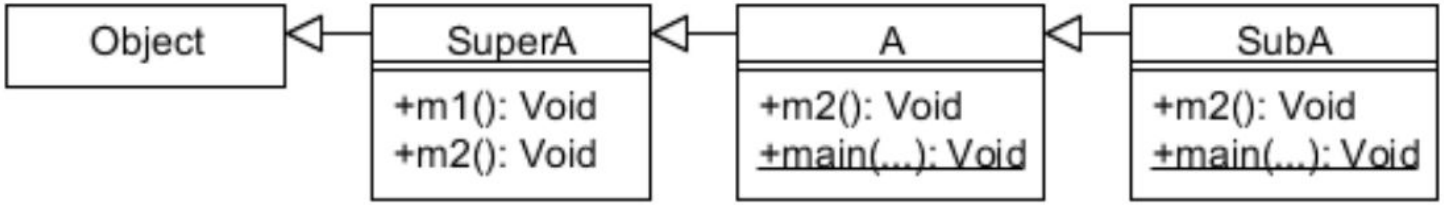
このthisがSuperAの場合

 # 1. コンパイル時

- thisの型: SuperA
- 引数の型 (): なし
- 探索範囲: Aおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: SuperA（探索の開始地点）
- 実行されるメソッド: SuperAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

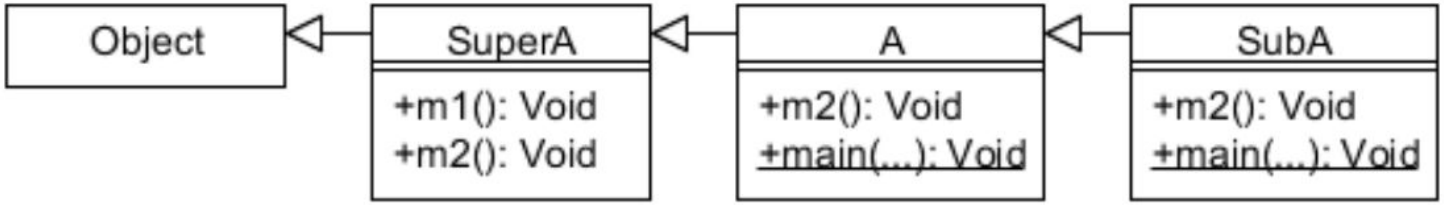
```
public void m2() { System.out.print("m2() in A;"); super.m2(); }
```

 # 1. コンパイル時

- superの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- superの型: SuperA（探索の開始地点）
- 実行されるメソッド: SuperAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public void m2() { System.out.print(“m2() in SubA;”); super.m2(); }
```

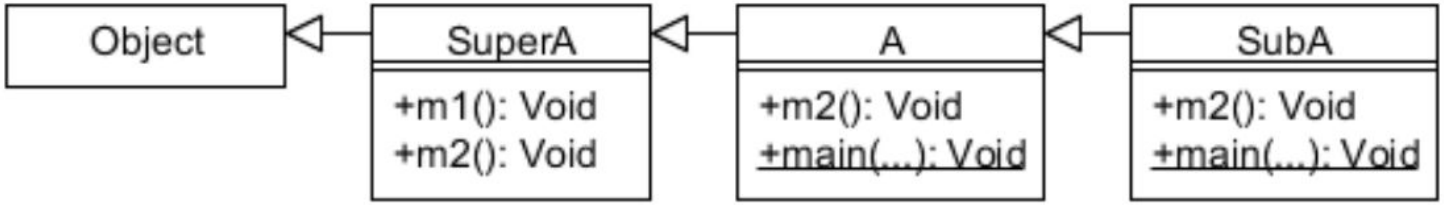
 # 1. コンパイル時

- superの型: A
- 引数の型 (): なし
- 探索範囲: Aおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- superの型: A（探索の開始地点）
- 実行されるメソッド: Aのm2()

p.14の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

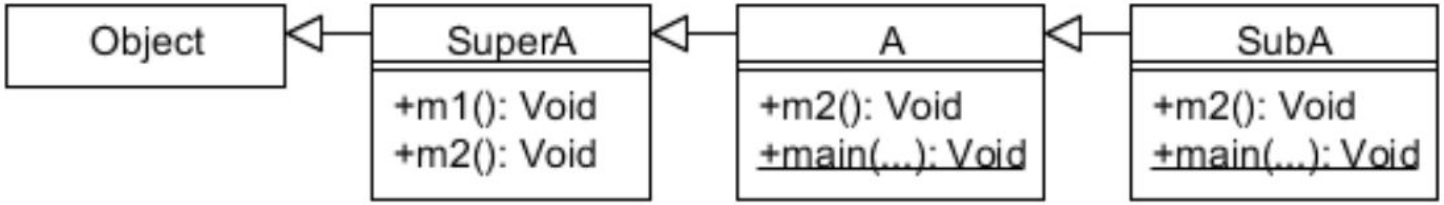
```
public static void main(String[] args) { (new A()).m1(); } // ← Aのmain(...)
public void m1() { System.out.print(“m1() in SuperA;”); this.m2(); }
```

 # 1. コンパイル時

- thisの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: A（探索の開始地点）
- 実行されるメソッド: Aのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

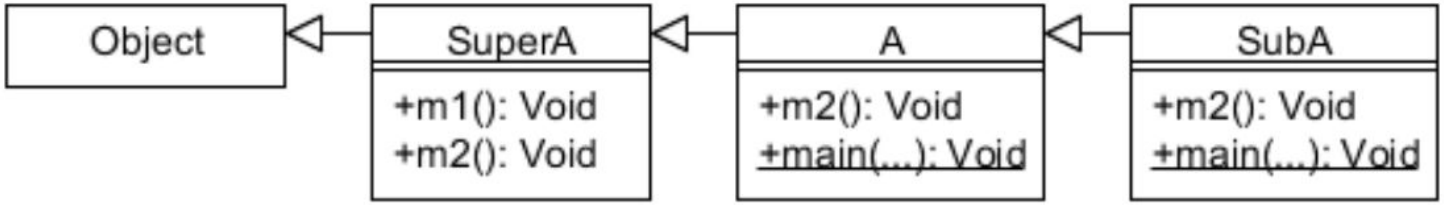
```
public static void main(String[] args) { (new A()).m1(); } // ← Aのmain(...)
public void m2() { System.out.print("m2() in A;"); super.m2(); }
```

 # 1. コンパイル時

- superの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- superの型: SuperA（探索の開始地点）
- 実行されるメソッド: SuperAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public static void main(String[] args) { (new A()).m1(); } // ← Aのmain(...)
public void m2() { System.out.print(“m2() in SuperA;”); }
```

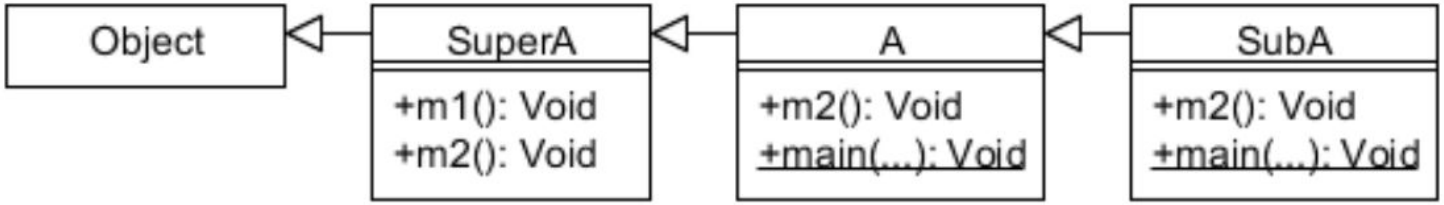
 # 1. コンパイル時

- SuperA，A，SubAのメソッド呼び出しなし

 # 2. 実行時

- SuperA，A，SubAのメソッド呼び出しなし
  （“m2() in SuperA;”をプリントして終了）

p.15の例



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

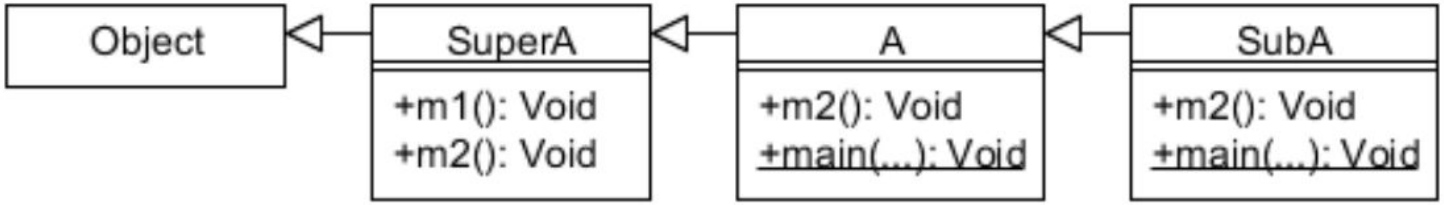
```
public static void main(String[] args) { (new SubA()).m1(); } // ← SubAのmain(...)
public void m1() { System.out.print("m1() in SuperA;"); this.m2(); }
```

 # 1. コンパイル時

- thisの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- thisの型: SubA（探索の開始地点）
- 実行されるメソッド: SubAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

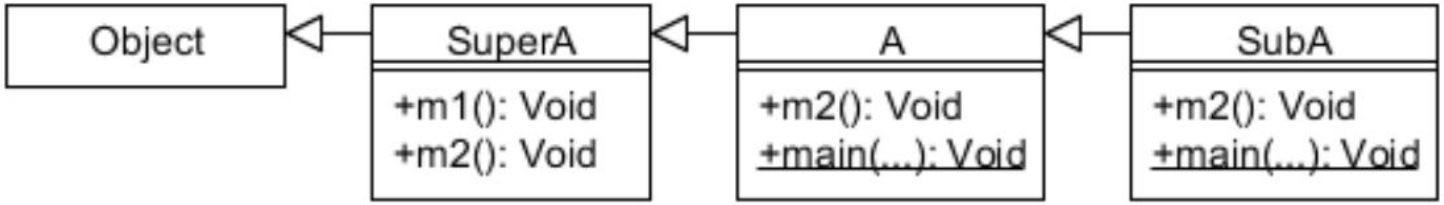
```
public static void main(String[] args) { (new SubA()).m1(); } // ← SubAのmain(...)
public void m2() { System.out.print("m2() in SubA;"); super.m2(); }
```

 # 1. コンパイル時

- superの型: A
- 引数の型 (): なし
- 探索範囲: Aおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- superの型: A（探索の開始地点）
- 実行されるメソッド: Aのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

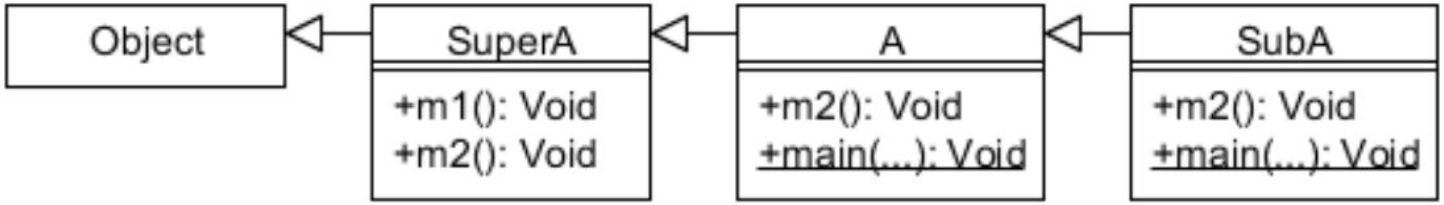
```
public static void main(String[] args) { (new SubA()).m1(); } // ← SubAのmain(...)
public void m2() { System.out.print("m2() in A;"); super.m2(); }
```

 # 1. コンパイル時

- superの型: SuperA
- 引数の型 (): なし
- 探索範囲: SuperAおよびそのすべての親クラス
- 候補: m2()
- 付与されるメソッドシグネチャ: m2()

 # 2. 実行時

- superの型: SuperA（探索の開始地点）
- 実行されるメソッド: SuperAのm2()



上図を前提に、以下のコードについてメソッド呼び出しがどのように解決されるかを述べる。

```
public static void main(String[] args) { (new SubA()).m1(); } // ← SubAのmain(...)
public void m2() { System.out.print(“m2() in SuperA;”); }
```

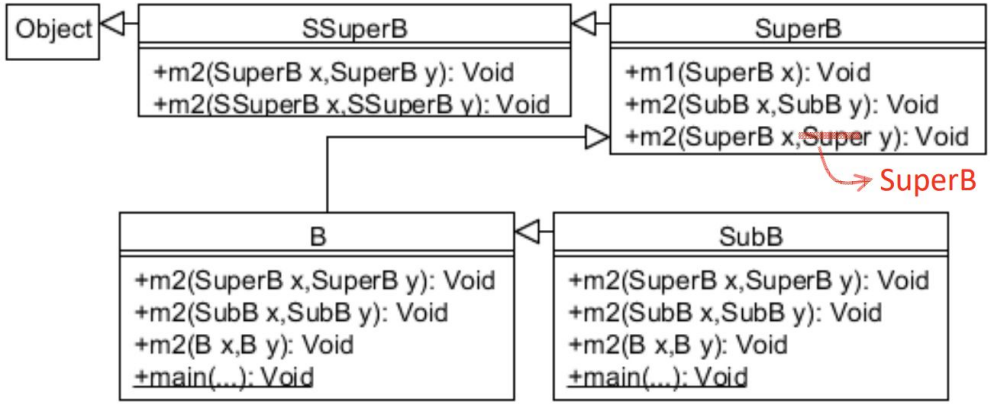
 # 1. コンパイル時

- SuperA，A，SubAのメソッド呼び出しなし

 # 2. 実行時

- SuperA，A，SubAのメソッド呼び出しなし
  （“m2() in SuperA;”をプリントして終了）

p.16以降の例の前提



左のクラス図で示されるクラスを実装するコードが、それぞれ下記の通りであることを前提に、クラスSuperB内の*AMsgPassing*が次ページ以降のコードであった場合に、メソッド呼び出しがどのように解決されるかを述べる。

In SSuperB:

```
m2(SuperB x,SuperB y)
{ …println("m2(SuperB x,SuperB y) in SSuperB;"); }
m2(SSuperB x,SSuperB y)
{ …println("m2(SSuperB x,SSuperB y) in SSuperB;"); }
```

In SuperB:

```
m1(SuperB x) { …print("m1(SuperB x) in SuperB;"); AMsgPassing }
m2(SubB x,SubB y) { …println("m2(SubB x,SubB y) in SuperB;"); }
m2(SuperB x,SuperB y)
{ …println("m2(SuperB x,SuperB y) in SuperB;"); }
```

In B:

```
m2(SuperB x,SuperB y) { …print("m2(SuperB x,SuperB y) in B;");
   super.m2(new B(),new B());  }
m2(SubB x,SubB y) { …print("m2(SubB x,SubB y) in B;");
   super.m2(new B(),new B());  }
m2(B x,B y) { …print("m2(B x,B y) in B;");  super.m2(new B(),new B());  }
main(String[] args) { (new B()).m1(new B()); }
```

In SubB:

```
m2(SuperB x,SuperB y) { …print("m2(SuperB x,SuperB y) in SubB;");
   super.m2(x,y);  }
m2(SubB x,SubB y) { …print("m2(SubB x,SubB y) in SubB;");
   super.m2(x,y);  }
m2(B x,B y) { …print("m2(B x,B y) in SubB;");
   super.m2(x,y);  }
main(String[] args) { (new SubB()).m1(new B()); }
```

p.19の例

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

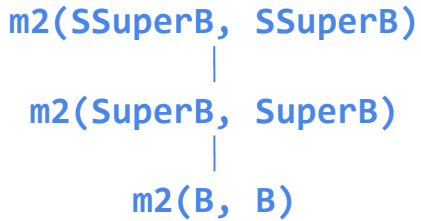
```
(new SubB()).m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

- (new SubB())の型: SubB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SubBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB), m2(B, B)
- 付与されるメソッドシグネチャ: m2(B, B)
  （最小要素であるため）



 # 2. 実行時

- (new SubB())の型: SubB（探索の開始地点）
- 実行されるメソッド: SubBのm2(B, B)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

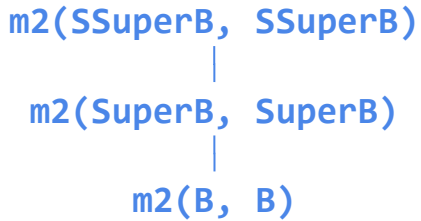
```
(new SubB()).m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

- (new SubB())の型: SubB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SubBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB), m2(B, B)
- 付与されるメソッドシグネチャ: m2(B, B)
  （最小要素であるため）



 # 2. 実行時

- (new SubB())の型: SubB（探索の開始地点）
- 実行されるメソッド: SubBのm2(B, B)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

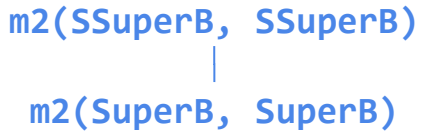
```
this.m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  （最小要素であるため）



 # 2. 実行時

- thisの型: B（探索の開始地点）
- 実行されるメソッド: Bのm2(SuperB, SuperB)

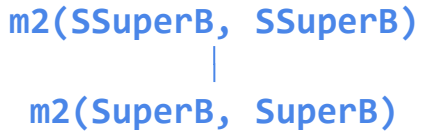
p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

```
this.m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時



- thisの型: SuperB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  （最小要素であるため）

 # 2. 実行時

- thisの型: SubB（探索の開始地点）
- 実行されるメソッド: SubBのm2(SuperB, SuperB)

p.20の例

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

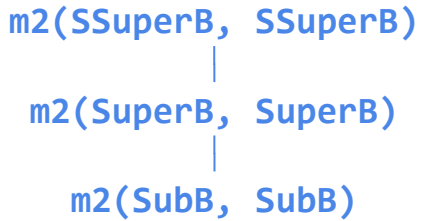
```
this.m2(new SubB(), new SubB());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (new SubB(), new SubB()): (SubB, SubB)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB), m2(SubB, SubB)
- 付与されるメソッドシグネチャ: m2(SubB, SubB)
  （最小要素であるため）



 # 2. 実行時

- thisの型: B（探索の開始地点）
- 実行されるメソッド: Bのm2(SubB, SubB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

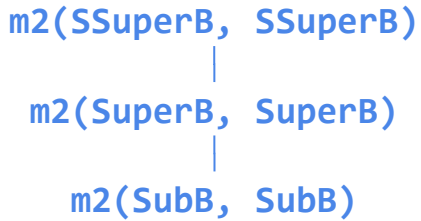
```
this.m2(new SubB(), new SubB());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (new SubB(), new SubB()): (SubB, SubB)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB), m2(SubB, SubB)
- 付与されるメソッドシグネチャ: m2(SubB, SubB)
  (最小要素であるため)



 # 2. 実行時

- thisの型: SubB(探索の開始地点)
- 実行されるメソッド: SubBのm2(SubB, SubB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

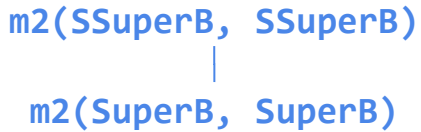
```
this.m2(this, this);
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (this, this): (SuperB, SuperB)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  （最小要素であるため）



 # 2. 実行時

- thisの型: B（探索の開始地点）
- 実行されるメソッド: Bのm2(SuperB, SuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

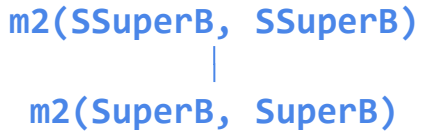
```
this.m2(this, this);
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (this, this): (SuperB, SuperB)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  （最小要素であるため）



 # 2. 実行時

- thisの型: SubB（探索の開始地点）
- 実行されるメソッド: SubBのm2(SuperB, SuperB)

p.21の例

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

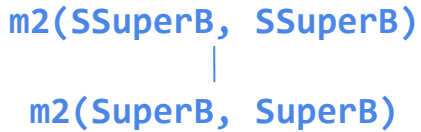
```
SuperB o = new SubB(); o.m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

- oの型: SuperB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  (最小要素であるため)



 # 2. 実行時

- oの型: SubB(探索の開始地点)
- 実行されるメソッド: SubBのm2(SuperB, SuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

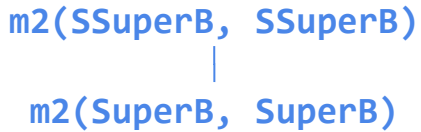
```
SuperB o = new SubB(); o.m2(new B(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

- oの型: SuperB
- 引数の型 (new B(), new B()): (B, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB), m2(SuperB, SuperB)
- 付与されるメソッドシグネチャ: m2(SuperB, SuperB)
  (最小要素であるため)



 # 2. 実行時

- oの型: SubB(探索の開始地点)
- 実行されるメソッド: SubBのm2(SuperB, SuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

```
SuperB o1 = new SubB(); o1.m2(new SSuperB(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

`m2(SSuperB, SSuperB)`

- o1の型: SuperB
- 引数の型 (new SSuperB(), new B()): (SSuperB, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB)
- 付与されるメソッドシグネチャ: m2(SSuperB, SSuperB)
  （最小要素であるため）

 # 2. 実行時

- o1の型: SubB（探索の開始地点）
- 実行されるメソッド: SSuperBのm2(SSuperB, SSuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

```
SuperB o1 = new SubB(); o1.m2(new SSuperB(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

`m2(SSuperB, SSuperB)`

- o1の型: SuperB
- 引数の型 (new SSuperB(), new B()): (SSuperB, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB)
- 付与されるメソッドシグネチャ: m2(SSuperB, SSuperB)
  （最小要素であるため）

 # 2. 実行時

- o1の型: SubB（探索の開始地点）
- 実行されるメソッド: SSuperBのm2(SSuperB, SSuperB)

p.22の例

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

```
this.m2(new SSuperB(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java B
```

 # 1. コンパイル時

`m2(SSuperB, SSuperB)`

- thisの型: SuperB
- 引数の型 (new SSuperB(), new B()): (SSuperB, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB)
- 付与されるメソッドシグネチャ: m2(SSuperB, SSuperB)
  （最小要素であるため）

 # 2. 実行時

- thisの型: B（探索の開始地点）
- 実行されるメソッド: SSuperBのm2(SSuperB, SSuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りであり、

```
this.m2(new SSuperB(), new B());
```

次のように実行した時に、メソッド呼び出しがどのように解決されるかを述べる。

```
% java SubB
```

 # 1. コンパイル時

`m2(SSuperB, SSuperB)`

- thisの型: SuperB
- 引数の型 (new SSuperB(), new B()): (SSuperB, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: m2(SSuperB, SSuperB)
- 付与されるメソッドシグネチャ: m2(SSuperB, SSuperB)
  （最小要素であるため）

 # 2. 実行時

- thisの型: SubB（探索の開始地点）
- 実行されるメソッド: SSuperBのm2(SSuperB, SSuperB)

p.39を前提に、クラスSuperB内の*AMsgPassing*が以下の通りである場合、

```
this.m2(new Object(), new B());
```

コンパイルは失敗する。

 # 1. コンパイル時

- thisの型: SuperB
- 引数の型 (new Object(), new B()):(Object, B)
- 探索範囲: SuperBおよびそのすべての親クラス
- 候補: なし
- 付与されるメソッドシグネチャ: なし
  （候補がないため）
  したがって、コンパイルエラーとなる。

 # 2. 実行時

- コンパイルエラーとなるため実行不可能である。