オブジェクト指向プログラミング
期末試験

問題冊子

「はじめ」の合図があるまで
問題冊子を開いてはいけません。

・この問題冊子は、（表紙を除いて）15ページあります。

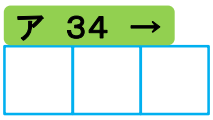
・この問題冊子は、10章に分かれており、各章に複数の問題があります。

・問題冊子に記載されているプログラムは、
全てプログラミング言語「Java」のプログラムです。

・各章の図とプログラムは、章ごとに独立しています。
また、各章のプログラムは、章ごとに完結しており、
他の章のプログラム（やその他のプログラム）に依存せずに独動作します。

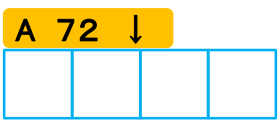
解答方法の例

例１）



カタカナの解答用紙に記入してください。
記入場所は、34番です。
記入方向は、横方向です。
記入する文字数は、3文字です。

例２）



アルファベットの解答用紙に記入してください。
記入場所は、72番です。
記入方向は、縦方向です。
記入する文字数は、4文字です。

1 基礎知識

●UMLの構成要素

(1) ソフトウェアの目的を表すユースケース図

(2) ソフトウェアの静的構造を表す ア 15 → [　][　][　][　][　][　][　] ダイヤグラム

(3) ソフトウェアの動的振る舞いを表す ア 11 → [　][　][　][　][　][　] ダイヤグラム

●継承と委譲の比較

	継承	委譲
ア 18 ↓ [　][　][　][　]	容易	やや複雑
柔軟性	ア 20 ↓ [　][　][　]	ア 7 → [　][　][　]

●インタフェースの特徴

(1) インタフェースを利用して、異なるクラスを同じものとして扱うことができる。

(2) インタフェースを利用して、ソフトウェアの ア 22 ↓ [　][　][　] を簡単に変更できる。

(3) インタフェースを利用して、必要な ア 22 ↓ [　][　][　] だけを公開することができる。

●デザインパターン

(1) E.Gamma, Helm.R, R.Johnson, J.Vissides,の４人が著書

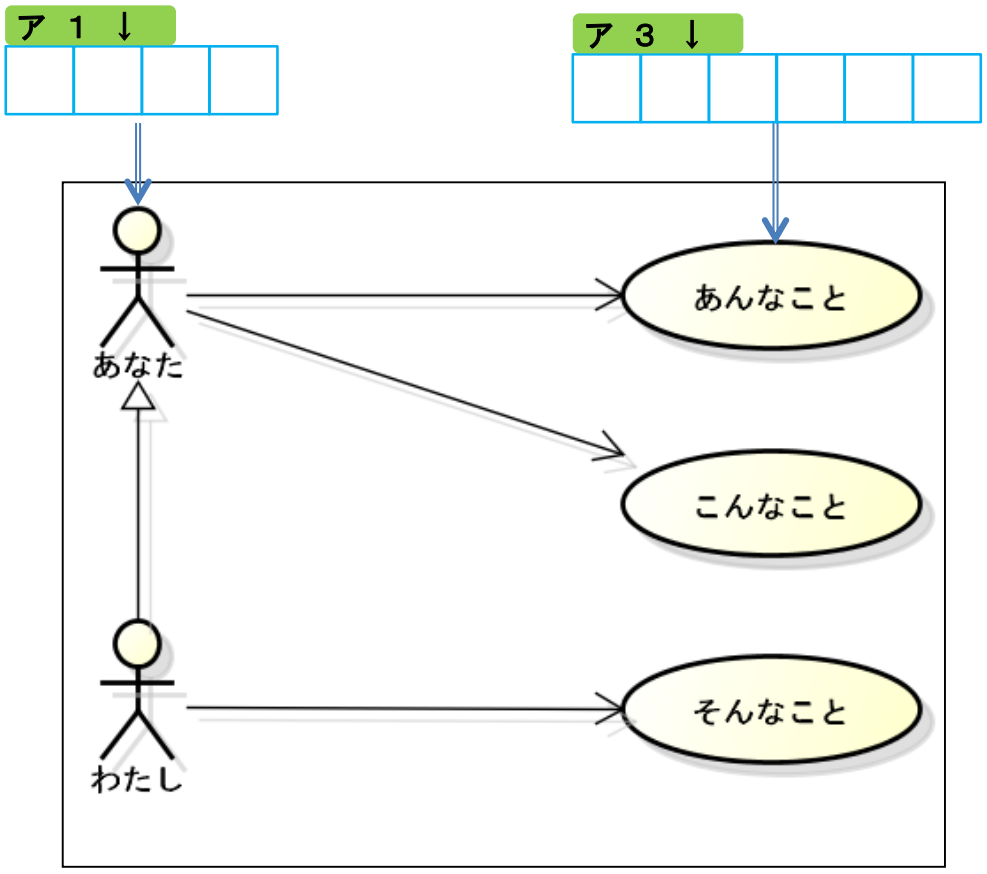
「Design Patterns」で A 30 → [　][　] 個のデザインパターンを提示した。

(2) デザインパターンは ア 25 → [　][　][　] に依存しない。

(3) どのデザインパターンにも長所と短所がある。

2 わたしにできること

●ユースケース図の構成要素の名称

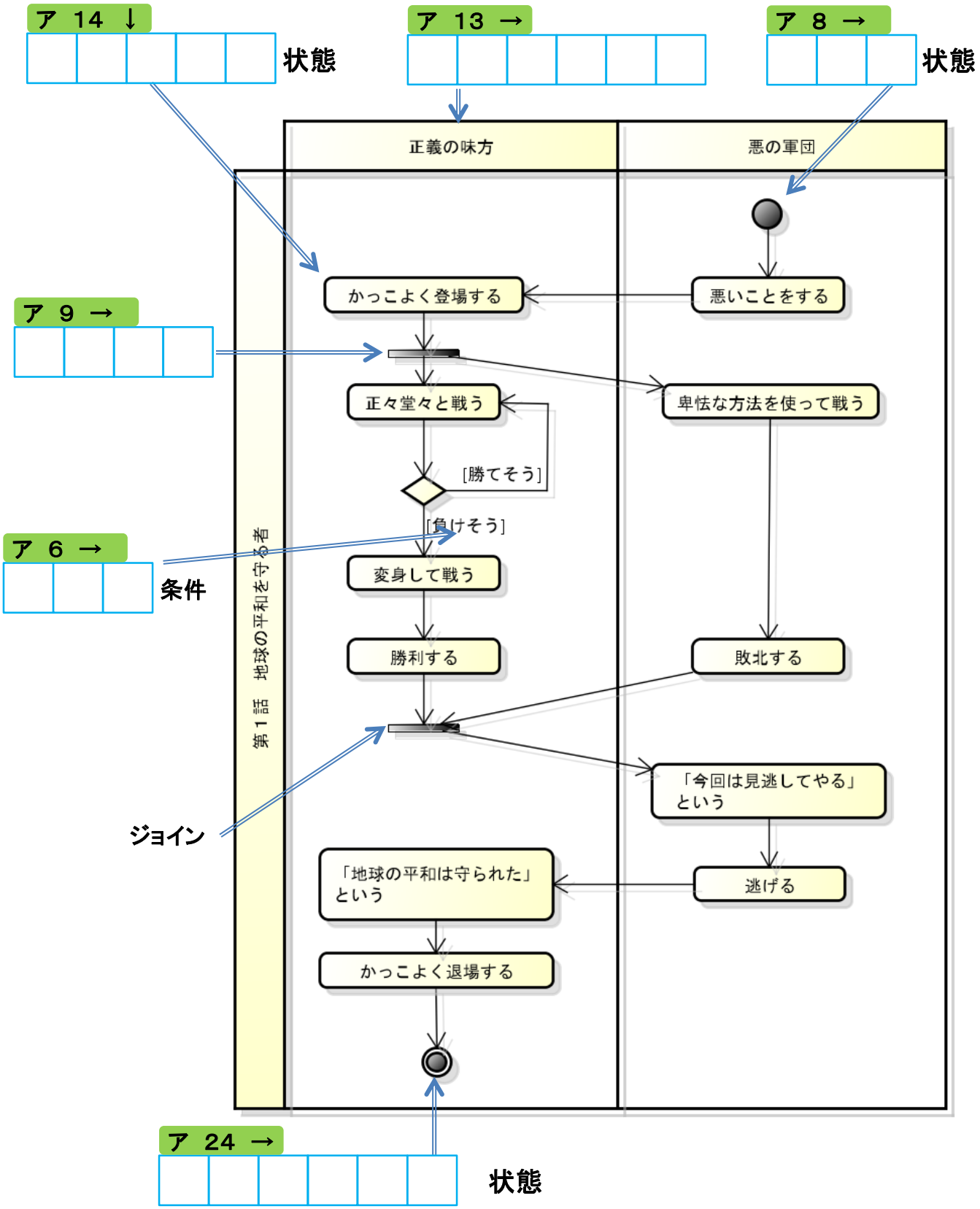


●上のユースケース図からわかること

「わたし」は A 22 ↓ 個のユースケースを利用できる。

3 正義の味方

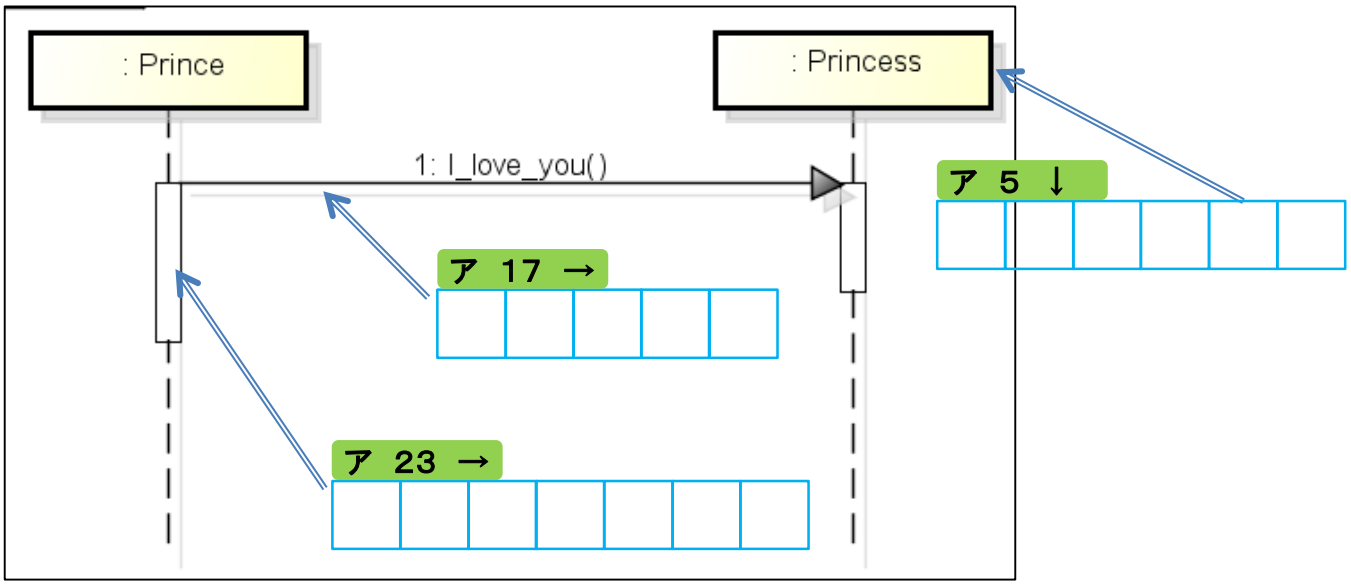
●アクティビティ図の構成要素の名称



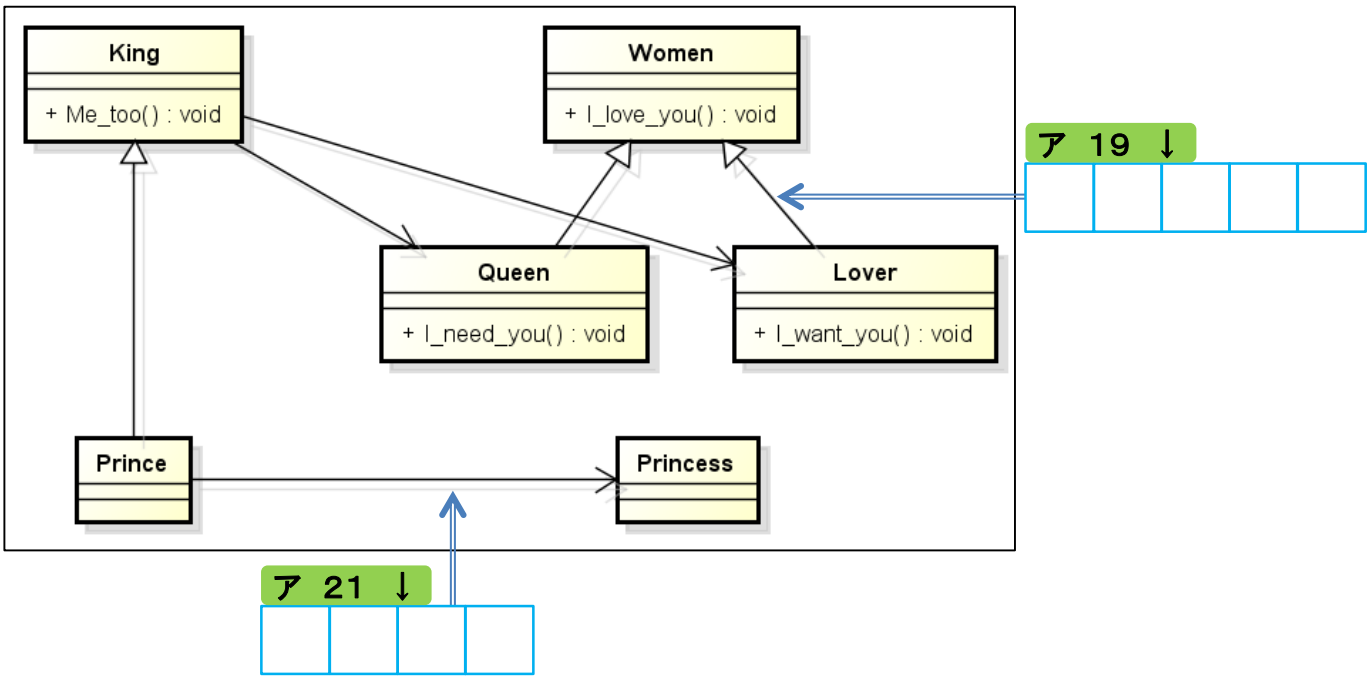
4 Princessの母親は。。。

●シーケンス図とクラス図の構成要素の名称

シーケンス図



クラス図



●上のシーケンス図とクラス図からわかること

上のクラス図では、Princessの親クラスに関する記述が抜けている。

Princessの親クラス（＝スーパークラス）は、A 35 → である。

5 ママのネックレス

プログラム　プログラムが正しく動作するようにすること

```
Public [A 23 ↓ 5 boxes] Son {
   RobotBrother doraemon;
   RobotSister dorami;
   public String search2(){
      return doraemon.[A 37 → 7 boxes]( );
   }
   public String search3(){
      return dorami.[A 28 ↓ 7 boxes]( );
   }
}
```

```
public class RobotBrother {
   public String searchB( ){   return "マサイ族の首飾り"; }
}
```

```
public class RobotSister{
   public String searchS( ){ return “素敵なネックレス”;  }
}
```

```
public class Mama {
   public static void main(String[ ] args ) {
      Son nobita = new Son( );
      nobita.doraemon = new RobotBrother( );
      nobita.dorami = new RobotSister( );
      String necklace = "";
      System.out.println("ママのネックレスをどこへやったの！");
      necklace = nobita.[A 21 → 7 boxes]();
      System.out.println(“ママをばかにしてるの！！");
      necklace = nobita.[A 18 ↓ 7 boxes]();
      System.out.println("これよ。。。やればできるじゃない。");
   }
}
```

6 地獄に現れた最強の男

プログラム プログラムが正しく動作するようにすること

```
public class Shinken {
   public String attack( )   {  return "北斗神拳";  }
   public String message( ){  return "北斗神拳は一子相伝";  }
}
```

```
public class Tokiken extends Shinken {
   public String attack( )   {  return "北斗有情拳";  }
   public String message( ){  return "安らかに死ぬがよい"; }
}
```

```
                                  A 25 ↓
public class Ryuuken extends [ | | | | | | ] {
   public String attack( )   {  return "七星点心";  }
}
```

```
                                  A 33 →
public class Siroken extends [ | | | | | | ] {
   public String message( ){  return "お前はもう死んでいる"; }
}
```

```
public class SaikyoNoOtoko{
   public static void main(String[] args) {
      Shinken keisyosya;
                                             A 17 →
      keisyosya = new [ | | | | | | ] ( );
      System.out.println(  keisyosya.attack( ) );
      System.out.println(  keisyosya.message() );
                                             A 14 →
      keisyosya = new [ | | | | | | ] ( );
      System.out.println(  keisyosya.attack( ) );
      System.out.println(  keisyosya.message( ) );
   }
}
```

実行結果

```
七星点心
安らかに死ぬがよい
七星点心
おまえはもう死んでいる
```

7 Black Butler

プログラム　プログラムが正しく動作するようにすること

```
public   interface   BasicButler {
   public  String [        ]( );
}
```
A 7 →

```
public  class  BlackButler [        ] BasicButler{
   public String [        ]( ) {
      return "あくまで執事ですから。";
   }
}
```
A 4 ↓　A 7 →

```
public  class  WhiteButler [        ] BasicButler{
   public String [        ]( ) {
      return "わたしは執事ですから。";
   }
}
```
A 4 ↓　A 7 →

```
public  class  SonoshitsujiYuno {
   public  static  void  main( String[ ]  args ) {
      [        ] sebastian;
      sebastian = new WhiteButler( );
      sebastian = new [        ]( );
      System.out.println( [        ] .sayReply( ) );
   }
}
```
A 19 →　A 19 ↓　A 18 →

実行結果

ア 1 →
[   ] 執事ですから。

8 孤独の観測者

プログラム　プログラムが正しく動作するようにすること

```
public class Labomem {
  public String sayHello ( ){ return “せかいをだませ” ; }
}
```

```
public class Mayushi extends [          ] {
   public String [          ] ( ){ return “とぅっとぅるー” ; }
}
```

A 13 →　A 7 ↓

```
public class Daaluuu extends [          ] {
   public String [          ] ( ){ return “だが、ことわる” ; }
}
```

A 24 →　A 7 →

```
class SGate{
    public static void main(String[] args){
      LaboMem okkalin = new LaboMem( );
      System.out.println( okalin.sayHello( ) );

      LaboMem mayushi = new Mayushi( );
      System.out.println( mayushi. [          ] ( ) );
      LaboMem daaluuu =new Daaluuu( );
      System.out.println( daaluuu. [          ] ( ) );
    }
}
```

A 11 →　A 36 →

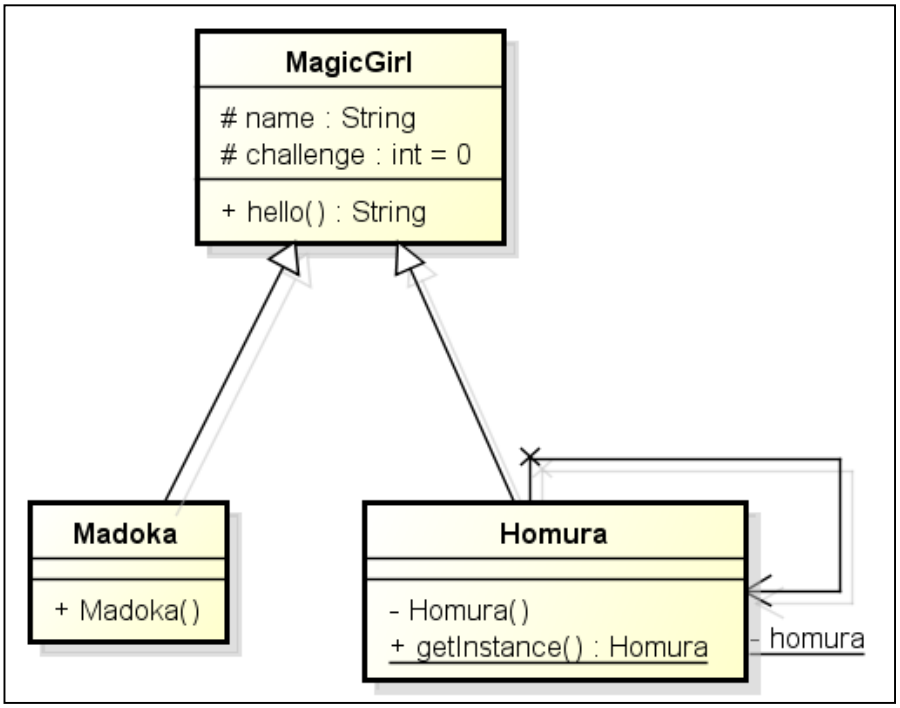
実行結果

```
せかいをだませ
[          ]
[          ]
```

ア 16 →　ア 10 ↓

9 もうだれにも頼らない(1/3)

クラス図



実行結果

【転校生を紹介します】

まどかです。(1 回目)

A 29 →

ほむらです。( ) 回目)

【転校生を紹介します】

まどかです。(1 回目)

A 30 ↓

ほむらです。( ) 回目)

【転校生を紹介します】

まどかです。(1 回目)

A 31 →

ほむらです。( ) 回目)

●このプログラムの説明

A 10 →

このプログラムでは、( ) パターンを利用している。

9 もう誰にも頼らない(2/3)

プログラム　※ヒント：クラス図を参照すること

```
public  class  MagicGirl {
   protected  String  name="";
   protected  int  challenge=0;
   public String hello(){  return  "はじめまして。"+name+"です。("+challenge+"回目)"; }
}
```

```
public  class  [A 5 →]  extends  MagicGirl {
   public  [A 5 →]  (){
      name="  [ア 2 ↓]  ";
      challenge++;
   }
}
```

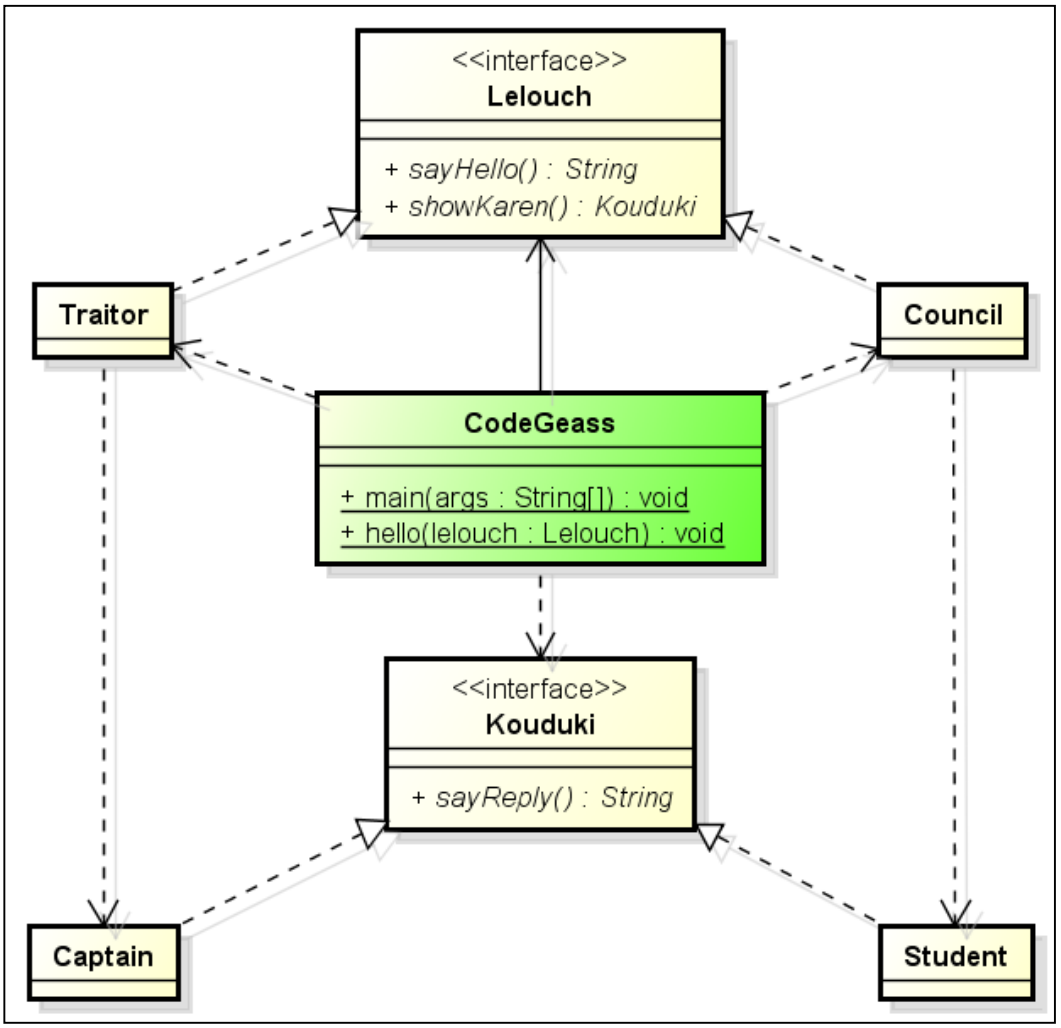
```
public  class  [A 12 →]  extends  MagicGirl{
   [A 8 →]  [A 15 →]  [A 12 →]
      [A 12 →]  =  [A 27 →]  [A 12 →]  ();
   [A 8 →]  [A 12 →]  (  ){
      name="  [ア 4 →]  ";
   }
   [A 15 →]  [A 12 →]
   public  getInstance(){
      [A 12 →]  .challenge++;
      return  ;  [A 12 →]
   }
}
```

9 もう誰にも頼らない(3/3)

```
public  class   MagicGirls {
   public   static   void   main( String[ ]  args ) {
      Madoka   madoka ;
      Homura   homura;

      madoka  =  new   Madoka( );
      homura  =  Homura.getInstance( );
      System.out.println(  "【転校生を紹介します】“ );
      System.out.println(  madoka.hello( )  );
      System.out.println(  homura.hello( )  );

      madoka  =  new   Madoka( );
      homura  =  Homura.getInstance( );
      System.out.println(  "【転校生を紹介します】“ );
      System.out.println(  madoka.hello( )  );
      System.out.println(  homura.hello( )  );

      madoka  =  new   Madoka( );
      homura  =  Homura.getInstance( );
      System.out.println(  "【転校生を紹介します】“ );
      System.out.println(  madoka.hello( )  );
      System.out.println(  homura.hello( )  );
   }
}
```

10 ゼロ(1/4)

クラス図



実行結果

【通常時】
ルルーシュ「生徒会副会長です。」
カレン「私、病弱で。。。」
【戦闘時】
ルルーシュ「わが名はゼロ！」
カレン「ゼロは私が守る！」

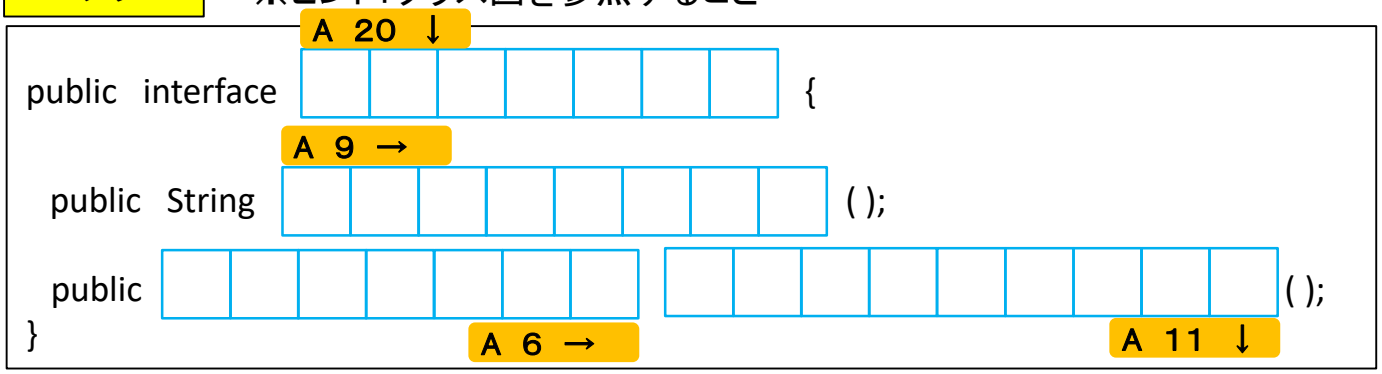
●このプログラムの説明

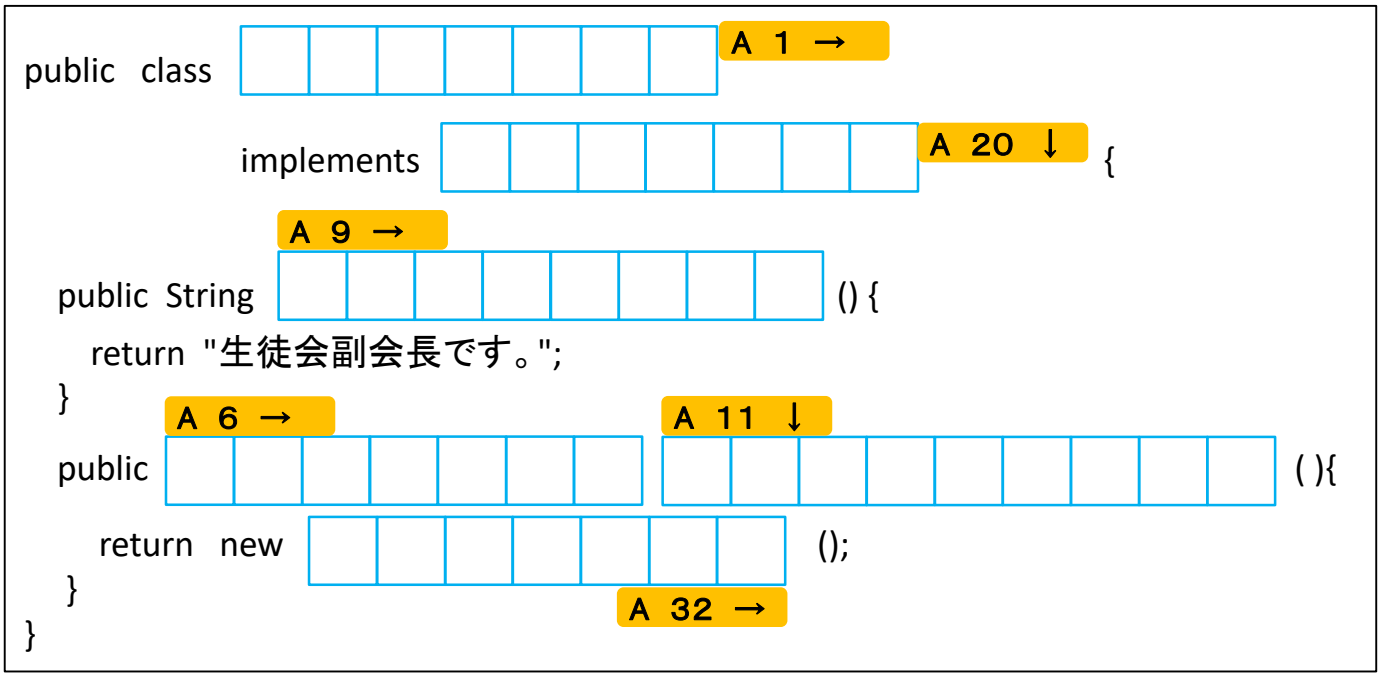
A 32 ↓

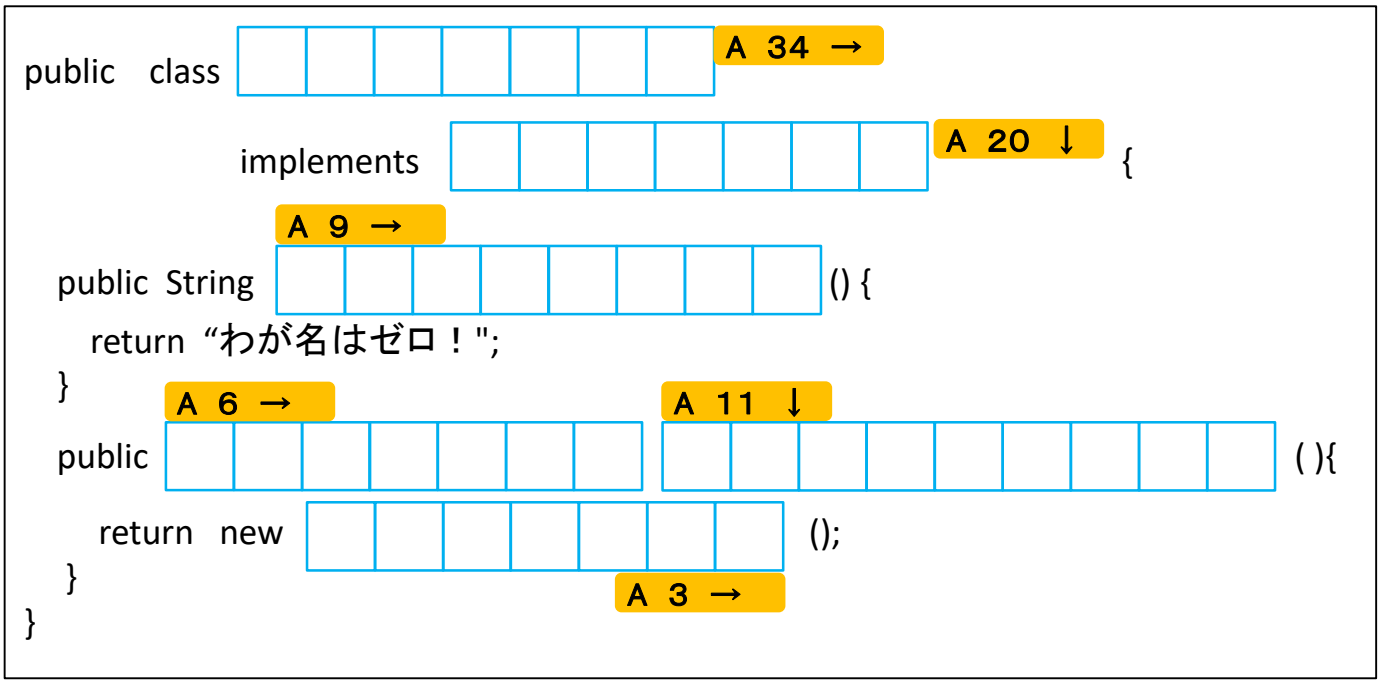
このプログラムでは [ ][ ][ ][ ][ ] パターンを利用している。

10 ゼロ(2/4)

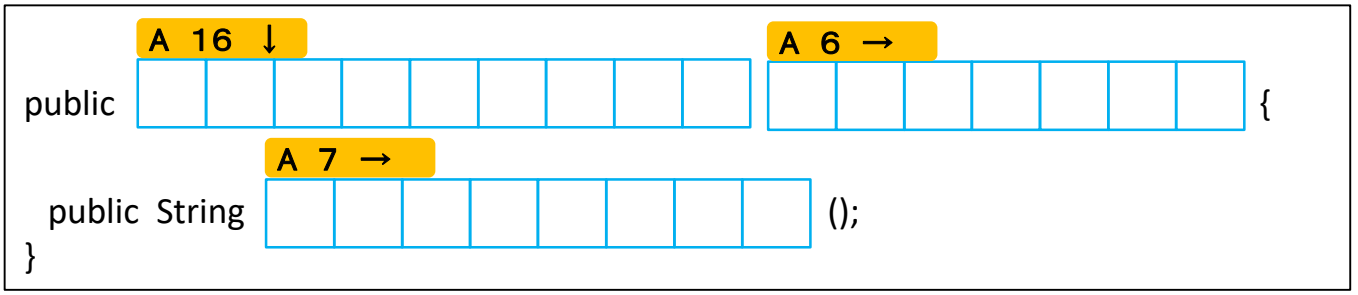
プログラム　※ヒント：クラス図を参照すること

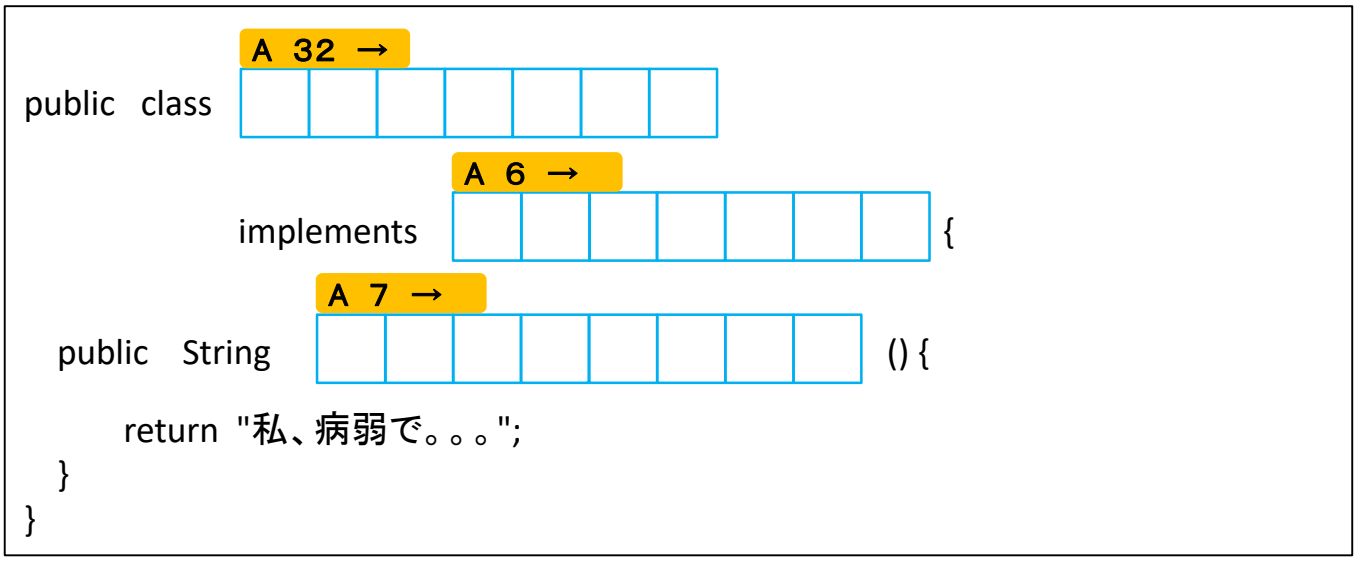


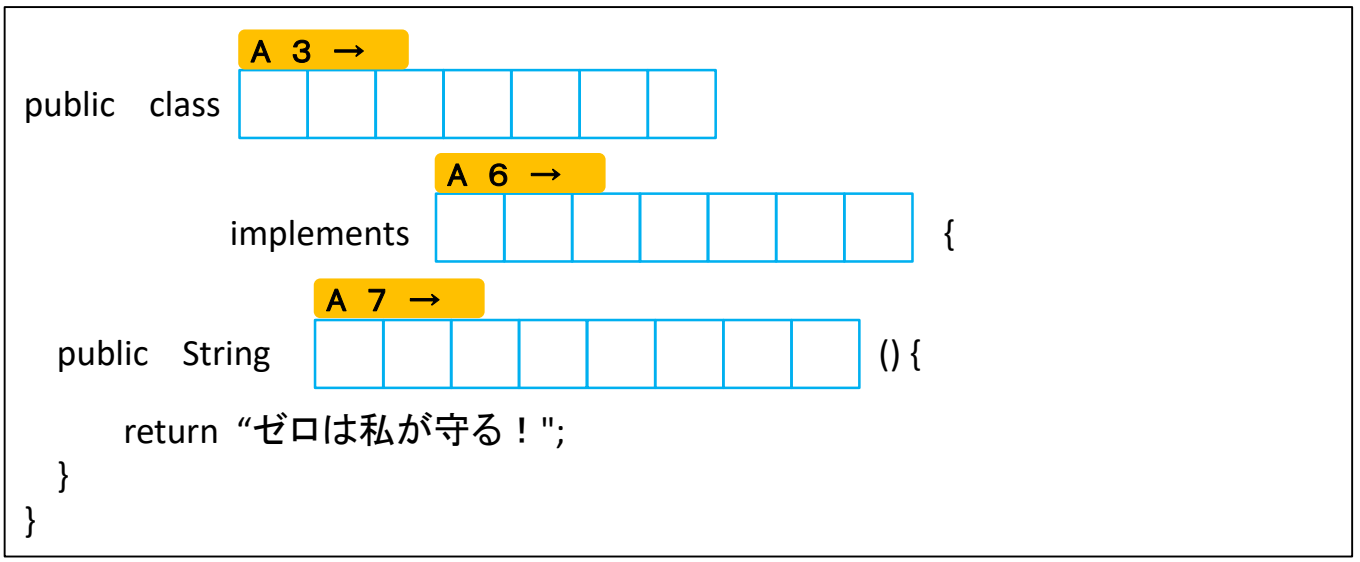




10 ゼロ(3/4)


A 16 ↓
A 6 →
public
{
A 7 →
public String
();
}

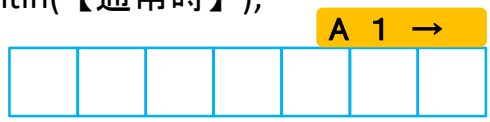

A 32 →
public class
A 6 →
implements
{
A 7 →
public String
() {
return "私、病弱で。。。";
}
}


A 3 →
public class
A 6 →
implements
{
A 7 →
public String
() {
return “ゼロは私が守る！";
}
}

10 ゼロ(4/4)

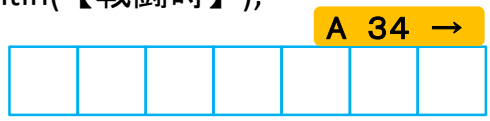
public class {

```
  public  static  void  main (String[] args){
    Lelouch lelouch;

    System.out.println("【通常時】");
```



lelouch = new ;

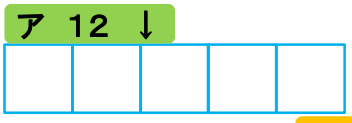
```
    hello ( lelouch );

    System.out.println("【戦闘時】");
```



lelouch = new ;

```
    hello ( lelouch );
  }

  public  static  void  hello ( Lelouch lelouch ){
```

System.out.println(“



”+

“「”+

.

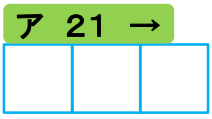
()+"」");

=

.

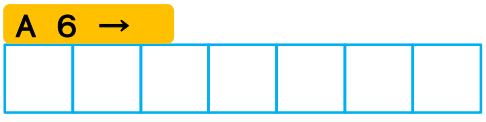
();

System.out.println(“



"+

“「”+



.

()+"」");

```
  }
}
```