オブジェクト指向設計　予習範囲・暗記事項・予習ノート・練習問題　for 講義 No.12

・暗記事項を参考にして予習ノートの作成
・予習範囲の熟読
・暗記事項の暗記
・教科書の演習問題を理解
上記を済ませてから練習問題に取り組んでください。
最終ページに解答を載せています。

予習範囲
7 章です.

暗記事項
【暗記事項 1】から【暗記事項 14】までは，すべて、「A: 正しい」が正答となります。予習確認テストでは表現を変えて「B: 間違っている」が答えになることもあります.
ただ覚えるだけでなく，教科書中の当該事項を説明した文章を読んで理解につとめてください。

【暗記事項1】親クラスで，具体的な処理が書けないメソッドがあるときは，その親クラスを抽象クラスとすることができる
【暗記事項2】具体的な処理が書けないメソッドを抽象メソッドという.
【暗記事項3】抽象クラスの定義の1行目では「アクセス修飾子　abstract　class　クラス名　{」と記述する.
【暗記事項4】抽象メソッドのメソッド定義は，「アクセス修飾子　abstract　戻り値の型　メソッド名(引数リスト);」と記述する
【暗記事項5】抽象メソッドのメソッド定義では，処理を記述せずにセミコロンで終わる.
【暗記事項6】修飾子final がメソッド定義に付けられた場合，子クラスでそのメソッドをオーバーライドできないことを意味している.
【暗記事項7】修飾子final がフィールド宣言に付けられた場合，その変数の値が変更できないこと（定数）を意味している.
【暗記事項8】抽象クラスを継承し，抽象メソッドの具体的な処理を記述した子クラスのことを具象クラスという.
【暗記事項9】抽象クラスを継承したが，一部の抽象メソッドについて具体的な処理を記述せずに放置した場合，この子クラスは抽象クラスのままであるので，キーワードabstractを用いてクラス定義を行わないといけない.
【暗記事項10】抽象クラスは参照型変数を宣言することができる.
【暗記事項11】抽象クラスはnew演算子によりオブジェクトを生成することができない.
【暗記事項12】抽象クラスの参照型変数は具象クラスのオブジェクトを参照できる
【暗記事項13】抽象クラスのクラス図では，クラス名，抽象メソッドを斜体にする
【暗記事項14】クラス図では抽象クラスの継承であっても通常の継承の矢印 ——▷ を用いる

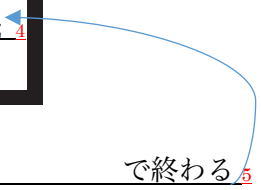
7.2 節　抽象クラス

抽象クラス
　　親クラスで，具体的な処理が書けないメソッドがあるときは，
　　その親クラスを　　クラスとすることができる [1]

抽象メソッド
　　具体的な　　が書けないメソッド [2]

抽象クラス

・クラスの定義方法：
　アクセス修飾子　　class クラス名 { [3]
　　クラスの中身
　}
・抽象メソッド：
　アクセス修飾子　　戻り値の型 メソッド名(引数リスト); [4]



　　抽象メソッドのメソッド定義では，処理を記述せずに　　で終わる [5]

修飾子 final
　　・メソッド定義に付けられた場合，
　　　子クラスでそのメソッドが　　できないことを表している [6]

　　・フィールド宣言に付けられた場合，
　　　その変数の値が変更できない，つまり　　を表している [7]

具象クラス
　　抽象クラスを継承し，抽象メソッドの具体的な処理を記述した子クラスのこと [8]

子クラスで抽象メソッドを実装しない場合
　　抽象クラスを継承した子クラスで，一部の抽象メソッドについて，具体的な処理を記述しなかった場合，
　　この子クラスは，抽象クラスのままであるので，キーワード abstract を用いてクラス定義を行わないと
　　いけない [9]

7.3 節　抽象クラスの参照型変数とオブジェクト

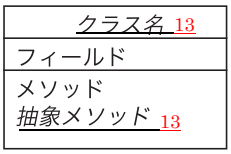
抽象クラスの参照型変数

・抽象クラスは　　を宣言することができる [10]
・new 演算子を用いてオブジェクトを　　できない [11]
・　クラス（　　クラス）のオブジェクトを参照できる [12]

7.4 節　抽象クラスのクラス図

抽象クラスのクラス図

・ソースコード中でキーワード abstract を付けていた箇所を斜体にする
・他の書式は通常のクラス図と同じ



抽象メソッドではないメソッドは斜体にしない

クラス図では，抽象クラスの継承であっても，通常の継承の矢印を用いる 14

練習問題
確認テストも，この練習問題と同じ方法で，解答してください．最終ページに答えを載せています．

解答が複数ある場合は、ハイフン で繋いで答えること。
　　例: A と B と C を解答したい場合，A-B-C と解答欄に記入する。答えがない場合は-1 と解答すること。
行番号を解答するとき、左詰めの 0 は取ること
　　例: 001 行目を解答するときは 1 を解答すること
該当する答えがないときは -1 と答えること

以下のエラーを検出する問題を出します：
・参照できないオブジェクトの参照
・呼び出せないメソッドの呼び出し
・抽象メソッドのサブクラス・インタフェースの実装クラスで，抽象メソッドを実装していない
・親クラスのコンストラクタが未定義・または，引数の型が合わないことによるエラー
・抽象クラスまたはインタフェースのオブジェクトを生成しようとしている．

それぞれの項目について簡単に説明します：
・参照できないオブジェクトの参照
　・継承による先祖-子孫関係であれば，先祖の参照型変数は，子孫のオブジェクトを参照できます．
　・子孫の参照型変数が，先祖のオブジェクトを参照できません．
　・インタフェースの参照型変数は，実装クラスのオブジェクトを参照できます．

・呼び出せないメソッドの呼び出し
　・例え参照型変数が子孫クラスや実装クラスのオブジェクトを参照していても，キャストをしない限り，
　　呼び出すことができるのは，参照型変数の型（クラス）で定義されたメソッドだけです．

・抽象クラスのサブクラス・インタフェースの実装クラスで，抽象メソッドを実装していない
　・抽象クラスのサブクラス，インタフェースの実装クラスでは，抽象メソッド（セミコロン『；』で
　　終わっていて，中身を定義していないメソッド）の処理を記述しないといけません．
　・ただし，抽象クラスのサブクラスが抽象クラスである場合，抽象メソッドを実装していなくてもいいです．

・親クラスのコンストラクタが未定義・または，引数の型が合わないことによるエラー
　・講義 No.11 で演習します（再履修の人は昨年の資料を参考にしてください）．

・抽象クラスまたはインタフェースのオブジェクトを生成しようとしている．
　・抽象クラスまたはインタフェースは処理を記述していないメソッドがあるため，オブジェクトを生成できま
　　せん．

System.out.println()（改行あり）は main メソッドにしかありません。
他の場所では、System.out.print()（改行なし）を使っています

行番号を振っていない空行に続いて、行番号を振りなおしているソースコードが続く場合は、
別の Java ファイル（クラス）であることを示しています。

本練習問題（前半）の答え方　この間違い探しの問題は 1 問しか出ませんが配点を高めに設定します

以下のソースコードについて【練習 01】を解答しなさい

```
001  public abstract class Parent {
002      private int field;
003      public Parent(int field) {
004          this.field = field;
005      }
006      public abstract void method();
007  }

011  public class Child extends Parent {
012      public void method2() {
013          System.out.println("hi");
014      }
015  }

021  public class TestDrive {
022      public static void main(String[] args) {
023          Child child = new Parent(3);
024      }
025  }
```

【練習 01】上記のソースコードで間違いがあれば，間違いの種類を下記の選択肢より選びなさい．
複数の間違いがある場合はハイフンでつないで解答すること

A-C-D

アルファベットの順に記述してください．エラーがなければ，-1 を記入してください．

それぞれ，23 行目，クラス Child，クラス Child（デフォルトコンストラクタは引数なしの親クラスのコンストラクタを呼び出します）で起きています

A: 参照できないオブジェクトの参照
B: 呼び出せないメソッドの呼び出し
C: 抽象クラスのサブクラス・インタフェースの実装クラスで，抽象メソッドを実装していない
D: 親クラスのコンストラクタが未定義・または，引数の型が合わないことによるエラー
E: 抽象クラスまたはインタフェースのオブジェクトを生成しようとしている．

以下のソースコードについて【練習 02】を解答しなさい

```
001  public abstract class Vehicle {
002      private String name;
003      public Vehicle(String name) {
004          this.name = name;
005      }
006  }

011  public interface IRun {
012      public void run();
013  }
014

021  public class Car extends Vehicle implements IRun {
022      public Car() {
023          super(3);
024      }
025      public void runnable() {
026          System.out.println("hi2");
027      }
028  }

031  public class VehicleTestDrive {
032      public static void main(String[] args) {
033          Vehicle vehicle = new Vehicle("Hello");
034          vehicle.runnable();
035      }
036  }
```

【練習 02】上記のソースコードで間違いがあれば，間違いの種類を下記の選択肢より選びなさい．
複数の間違いがある場合はハイフンでつないで解答すること

A: 参照できないオブジェクトの参照
B: 呼び出せないメソッドの呼び出し
C: 抽象クラスのサブクラス・インタフェースの実装クラスで，抽象メソッドを実装していない
D: 親クラスのコンストラクタが未定義・または，引数の型が合わないことによるエラー
E: 抽象クラスまたはインタフェースのオブジェクトを生成しようとしている．

以下のソースコードについて【練習 03】を解答しなさい

```
001  public abstract class PreCure {
002      protected String name;
003      abstract void transform();
004      abstract void attack();
005      public PreCure(String name) {
006          System.out.print("OO");
007          this.name = name;
008      }
009      public void battle(String enemy) {
010          System.out.print(enemy + "  が  あらわれた");
011          transform();
012          System.out.print(name + "  に  へんしんした");
013          attack();
014      }
015  }

021  public abstract class HappinessChargePreCure extends PreCure{
022      public HappinessChargePreCure(String name) {
023          super(name);
024      }
025      public void attack() {
026          System.out.print("プリキュア・ハピネスビッグバーン！！");
027      }
028  }

031  public class CureLovely extends HappinessChargePreCure {
032      public CureLovely() {
033          super("キュアラブリー");
034      }
035  }

041  public class CureFortune extends HappinessChargePreCure {
042      public void transform() {
043          System.out.print("夜空にきらめく希望の星! キュアフォーチュン!");
044      }
045  }

051  public class PreCureTestDrive {
052      public static void main(String[] args) {
053          PreCure precure = new CureLovely();
054          precure.attack();
055      }
056  }
```

【練習 03】上記のソースコードで間違いがあれば，間違いの種類を下記の選択肢より選びなさい．
複数の間違いがある場合はハイフンでつないで解答すること

A: 参照できないオブジェクトの参照
B: 呼び出せないメソッドの呼び出し
C: 抽象クラスのサブクラス・インタフェースの実装クラスで，抽象メソッドを実装していない
D: 親クラスのコンストラクタが未定義・または，引数の型が合わないことによるエラー
E: 抽象クラスまたはインタフェースのオブジェクトを生成しようとしている．

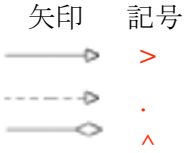
以下のソースコードについて【練習 04】から【練習 14】を解答しなさい

```
001  public abstract class A_CT12a {
002      protected String type;
003      public A_CT12a(String type) {
004          this.type = type;
005          System.out.print("00");
006      }
007      public void process() {
008          run();
009          System.out.print(type);
010      }
011      public String getType() {
012          System.out.print("01");
013          return type;
014      }
015      public abstract void run();
016  }

021  public abstract class B_CT12aC1 extends A_CT12a {
022      protected String name;
023      public B_CT12aC1(String name) {
024          super("02");
025          this.name = name;
026          System.out.print("03");
027      }
028      public String getName() {
029          System.out.print("04");
030          return name;
031      }
032  }

041  public class C_CT12aC2 extends A_CT12a {
042      public C_CT12aC2(String arg) {
043          super(arg);
044          System.out.print("05");
045      }
046      public C_CT12aC2() {
047          super("06");
048          System.out.print("07");
049      }
050      public void run() {
051          System.out.print("08");
052      }
053  }

061  public class D_CT12aC1C1 extends B_CT12aC1 {
062      public D_CT12aC1C1() {
063          super("07");
064          System.out.print("08");
065      }
066      public void run() {
067          System.out.print("09");
068      }
069  }

081  public class E_CT12aC1C2 extends B_CT12aC1 {
082      public E_CT12aC1C2() {
083          super("10");
084          System.out.print("11");
085      }
```

```
086         public void run() {
087             System.out.print("12");
088         }
089     }

101     public class F_CT12aC2C1 extends C_CT12aC2 {
102         public F_CT12aC2C1() {
103             System.out.print("12");
104         }
105         public void run() {
106             System.out.print("13");
107         }
108     }

111     public class G_CT12aC2C2 extends C_CT12aC2 {
112         public G_CT12aC2C2() {
113             System.out.print("14");
114         }
115     }

121     public class CT12TestDrive {
122         public static void main(String[] args) {
123             A_CT12a ct12a = new F_CT12aC2C1();
124             ct12a.process();
125             System.out.println("A");
126
127             ct12a = new E_CT12aC1C2();
128             ct12a.process();
129             System.out.println("B");
130
131             ct12a = new D_CT12aC1C1();
132             ct12a.process();
133             System.out.println("C");
134
135             ct12a = new G_CT12aC2C2();
136             ct12a.process();
137             System.out.println("D");
138         }
139     }
```

クラス図の解答でイタリックにする必要がある場合は『/』で挟んでください！

【練習 04】出力の一行目を答えなさい
【練習 05】出力の二行目を答えなさい
【練習 06】出力の三行目を答えなさい
【練習 07】出力の四行目を答えなさい
【練習 08】A_CT12a のクラス図の一行目を答えなさい
【練習 09】A_CT12a のクラス図の二行目を答えなさい
【練習 10】A_CT12a のクラス図の三行目を答えなさい
【練習 11】A_CT12a のクラス図の六行目を答えなさい
【練習 12】B_CT12aC1 のクラス図の一行目を答えなさい
【練習 13】B_CT12aC1 のクラス図の二行目を答えなさい



クラスの先頭の文字で，上の矢印を表す記号のどれかを囲みます．
矢印の先にあたるクラスが後ろに来るように答えること．

【練習 14】クラス間の関係を表す矢印を答えなさい

以下のソースコードについて【練習 15】から【練習 22】を解答しなさい

```
001  public interface IServiceType {
002      public String serve();
003  }
004

011  public abstract class ConsumerElectric {
012      private IServiceType iServiceType;
013      private String name;
014
015      public ConsumerElectric(String name, IServiceType iServiceType) {
016          this.name = name;
017          this.iServiceType = iServiceType;
018          System.out.print("a");
019      }
020
021      public abstract String job();
022
023      public void service() {
024          String service = iServiceType.serve();
025          String job = job();
026          System.out.print(name + "は" + service + job);
027      }
028
029      public String getName() {
030          System.out.print("b");
031          return name;
032      }
033  }

041  public class ByHand implements IServiceType {
042      public String serve() {
043          System.out.print("c");
044          return "手動で";
045      }
046  }

051  public class Electric implements IServiceType {
052      public String serve() {
053          System.out.print("d");
054          return "電動で";
055      }
056  }

061  public class F_Clearner extends ConsumerElectric {
062      public F_Clearner(String name, IServiceType iServiceType) {
063          super(name, iServiceType);
064          System.out.print("e");
065      }
066      public String job() {
067          System.out.print("f");
068          return "掃除します";
069      }
070  }
```

```
081  public class J_Broom extends F_Clearner {
082      public J_Broom() {
083          super("ほうき", new ByHand());
084          System.out.print("i");
085      }
086  }

091  public class VacuumSweeper extends F_Clearner {
092      public VacuumSweeper() {
093          super("掃除機", new Electric());
094          System.out.print("j");
095      }
096  }

101  public class Laundry extends ConsumerElectric {
102      public Laundry(String name, IServiceType iServiceType) {
103          super(name, iServiceType);
104          System.out.print("g");
105      }
106      public String job() {
107          System.out.print("h");
108          return "洗濯します";
109      }
110  }

121  public class WashBoard extends Laundry {
122      public WashBoard() {
123          super("洗濯板", new ByHand());
124          System.out.print("k");
125      }
126  }

131  public class X_WasherMachine extends Laundry {
132      public X_WasherMachine() {
133          super("洗濯機", new Electric());
134          System.out.print("l");
135      }
136  }

141  public class TestDrive {
142      public static void main(String[] args) {
143          ConsumerElectric v1 = new VacuumSweeper();
144          System.out.println("W");
145          v1.service();
146          System.out.println("X");
147          v1 = new WashBoard();
148          System.out.println("Y");
149          v1.service();
150          System.out.println("Z");
151      }
152  }
```

矢印	記号
	>
	.
	^

【練習 15】出力の一行目を答えなさい
【練習 16】出力の二行目を答えなさい
【練習 17】出力の三行目を答えなさい
【練習 18】出力の四行目を答えなさい
【練習 19】ConsumerElectric のクラス図の一行目を答えなさい
【練習 20】ConsumerElectric のクラス図の二行目を答えなさい
【練習 21】ConsumerElectric のクラス図の五行目を答えなさい
【練習 22】クラス間の関係を表す矢印を答えなさい

以下のソースコードについて【練習 23】から【練習 33】を解答しなさい

```
001  public interface IServiceType {
002      public String serve();
003  }

011  public abstract class Vehicle {
012      private IServiceType iServiceType;
013      private String name;
014      public Vehicle(String name, IServiceType iServiceType) {
015          this.name = name;
016          this.iServiceType = iServiceType;
017          System.out.print("a");
018      }
019
020      public abstract String move();
021
022      public void service() {
023          String service = iServiceType.serve();
024          String move = move();
025          System.out.print(name + "は" + service + move);
026      }
027      public String getName() {
028          System.out.print("b");
029          return name;
030      }
031  }

041  public class Cheap implements IServiceType {
042      public String serve() {
043          System.out.print("c");
044          return "少し窮屈に";
045      }
046  }

051  public class Laxury implements IServiceType {
052      public String serve() {
053          System.out.print("d");
054          return "とても快適に";
055      }
056  }

061  public class Airplane extends Vehicle {
062      public Airplane(String name, IServiceType iServiceType) {
063          super(name, iServiceType);
064          System.out.print("d");
065      }
066      public String move() {
067          System.out.print("e");
068          return "飛びます";
069      }
070  }

081  public class B_Car extends Vehicle {
082      public B_Car(String name, IServiceType iServiceType) {
083          super(name, iServiceType);
084          System.out.print("f");
085      }
086      public String move() {
087          System.out.print("g");
088          return "走ります";
089      }
090  }

101  public class D_ANA extends Airplane {
102      public D_ANA() {
```

```
103             super("ANA", new Laxury());
104             System.out.print("h");
105         }
106     }

111     public class F_LCC extends Airplane {
112         public F_LCC() {
113             super("LCC", new Cheap());
114             System.out.print("i");
115         }
116     }

121     public class EV extends B_Car {
122         public EV() {
123             super("電気自動車", new Laxury());
124             System.out.print("j");
125         }
126     }

131     public class Kei extends B_Car {
132         public Kei() {
133             super("軽自動車", new Cheap());
134             System.out.print("k");
135         }
136     }

141     public class TestDrive {
142         public static void main(String[] args) {
143             Vehicle v1 = new F_LCC();
144             System.out.println("W");
145             v1.service();
146             System.out.println("X");
147             v1 = new EV();
148             System.out.println("Y");
149             v1.service();
150             System.out.println("Z");
151         }
152     }
```

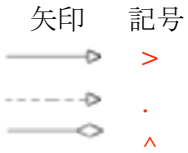
【練習 23】出力の一行目を答えなさい
【練習 24】出力の二行目を答えなさい
【練習 25】出力の三行目を答えなさい
【練習 26】出力の四行目を答えなさい
【練習 27】IServiceType のクラス図の一行目を答えなさい
【練習 28】IServiceType のクラス図の二行目を答えなさい
【練習 29】IServiceType のクラス図の三行目を答えなさい
【練習 30】Vehicle のクラス図の一行目を答えなさい
【練習 31】Vehicle のクラス図の二行目を答えなさい
【練習 32】Vehicle のクラス図の五行目を答えなさい

矢印	記号
	>
	.
	^

【練習 33】クラス間の関係を表す矢印を答えなさい

```
001  public interface A_Quality {
002        public String explain();
003  }

011  public abstract class B_SoundDevice {
012     protected A_Quality quality;
013     protected String name;
014     public B_SoundDevice() {
015        System.out.print("a");
016     }
017     public B_SoundDevice(String name,    A_Quality quality) {
018        this.name = name;
019        this.quality = quality;
020        System.out.print("b");
021     }
022     public abstract String job();
023     public void service() {
024        String service = quality.explain();
025        String job = job();
026        System.out.print(name + " " + job + "    " + service);
027     }
028     public String getName() {
029        System.out.print("b");
030        return name;
031     }
032  }

041  public class C_HighRes implements    A_Quality {
042       public String explain() {
043             System.out.print("d");
044             return "HQ";
045       }
046  }

051  public class D_Normal implements A_Quality {
052     public String explain() {
053        System.out.print("c");
054        return "NQ";
055     }
056  }

061  public class E_Headphone extends B_SoundDevice {
062     public E_Headphone(String name,    A_Quality quality) {
063        super(name, quality);
064        System.out.print("e");
065     }
066     public String job() {
067        System.out.print("f");
068        return "whisper";
069     }
070  }

081  public class F_Speaker extends B_SoundDevice {
082     public F_Speaker(String name, A_Quality quality) {
083        super(name, quality);
084        System.out.print("g");
085     }
086     public String job() {
087        System.out.print("h");
088        return "sound";
089     }
090  }
```

```
101  public class G_HighResPhone extends E_Headphone {
102      public G_HighResPhone() {
103          super("HRP", new C_HighRes());
104          System.out.print("i");
105      }
106  }

111  public class H_HighResSpeaker extends F_Speaker {
112      public H_HighResSpeaker() {
113          super("HRS", new C_HighRes());
114          System.out.print("k");
115      }
116  }

121  public class I_Normalphone extends E_Headphone {
122          public I_Normalphone() {
123              super("NP", new D_Normal());
124              System.out.print("j");
125          }
126  }

131  public class J_NormalSpeaker extends F_Speaker {
132      public J_NormalSpeaker() {
133          super("NS", new D_Normal());
134          System.out.print("l");
135      }
136  }

141  public class TestDrive {
142      public static void main(String[] args) {
143          B_SoundDevice v1 = new H_HighResSpeaker();
144          System.out.println("0");
145          v1.service();
146          System.out.println("1");
147          v1 = new J_NormalSpeaker();
148          System.out.println("2");
149          v1.service();
150          System.out.println("3");
151      }
152  }
```



【練習 34】出力の一行目を答えなさい
【練習 35】出力の二行目を答えなさい
【練習 36】出力の三行目を答えなさい
【練習 37】出力の四行目を答えなさい
【練習 38】B_SoundDevice のクラス図の一行目を答えなさい
【練習 39】B_SoundDevice のクラス図の二行目を答えなさい
【練習 40】B_SoundDevice のクラス図の四行目を答えなさい
【練習 41】B_SoundDevice のクラス図の五行目を答えなさい
【練習 42】B_SoundDevice のクラス図の六行目を答えなさい
【練習 43】クラス間の関係を表す矢印を答えなさい

【練習 02】B-C-D-E
（それぞれ，34 行目， クラス Car, 23 行目, 33 行目です）
【練習 03】C-D
（それぞれ，クラス CureLovely，クラス CureFortune のデフォルトコンストラクタです）
【練習 04】0007121306A
【練習 05】0003111202B
【練習 06】0003080902C
【練習 07】0007140806D
【練習 08】/A_CT12a/
【練習 09】# type: String
【練習 10】+ A_CT12a(type: String)
【練習 11】/+ run(): void/
【練習 12】/B_CT12aC1/
【練習 13】# name: String
【練習 14】B>A-C>A-D>B-E>B-F>C-G>C

【練習 15】aejW
【練習 16】df 掃除機は電動で掃除します X
【練習 17】agkY
【練習 18】ch 洗濯板は手動で洗濯します Z
【練習 19】/ConsumerElectric/
【練習 20】- iServiceType: IServiceType
【練習 21】/+job(): String/
【練習 22】B.I-E.I-F>C-I^C-J>F-L>C-V>F-W>L-X>L

【練習 23】adiW
【練習 24】ceLCC は少し窮屈に飛びます X
【練習 25】afjY
【練習 26】dg 電気自動車はとても快適に走ります Z
【練習 27】<<interface>>
【練習 28】IServiceType
【練習 29】+ serve(): String
【練習 30】/Vehicle/
【練習 31】- iServiceType: IServiceType
【練習 32】/+ move(): String/
【練習 33】A>V-B>V-C.I-D>A-E>B-F>A-I^V-K>B-L.I

【練習 34】bgk0

【練習 35】dhHRS sound　HQ1

【練習 36】bgl2

【練習 37】chNS sound　NQ3

【練習 38】/B_SoundDevice/

【練習 39】# quality: A_Quality

【練習 40】+ B_SoundDevice()

【練習 41】+ B_SoundDevice(name: String, quality: A_Quality)

【練習 42】/+ job(): String/

【練習 43】A^B-C.A-D.A-E>B-F>B-G>E-H>F-I>E-J>F