1-1. オブジェクトを作って動かす

1.（1）イ　（2）イ　（3）イ　（4）ア

2.

```
package pass2;
public class Dice52 {
     int    n;
     int    getN()     {return    n;}
     void   setN(int m)    {   n = m; }
     int    play(){
         n   =   (int)(Math.random()*52) + 1;
         return n;
     }
}
```

【解説】1～52 の整数乱数なので，Math.random()の値を 52 倍しておきます

3.問 1

```
package pass3;
public class Card {
     int    suit;       // 種類（0=スペード，1=ハート，2=クラブ，3=ダイヤ）
     int    number;     // 札番号（1～13）
     int    getSuit()      { return   suit; }        // 種類を返す
     int    getNumber()    { return   number; }      // 札番号を返す
     void   setSuit(int s)    {suit = s;}           // 種類をセットする
     void   setNumber(int m)  {number = m;}         // 札番号をセットする
     int    toSuit(int n)     {return (n-1)/13;}    // 種類に変換する
     int    toNumber(int n)   {return (n-1)%13 + 1;} // 札番号に変換する
}
```

問 2

```
package pass3;
public class Exec {
     public static void main(String[] args) {
         Dice52 d   =   new  Dice52();
         Card   c   =   new  Card();
         int    n   =   d.play();  // サイコロを振って 1～52 のどれかを得る
         c.setSuit(c.toSuit(n));  // サイコロの数を種類に変換してセット
         c.setNumber(c.toNumber(n)); // サイコロの数を札番号に変換してセット
         System.out.println(c.getSuit()+"/"+c.getNumber()); // 表示
     }
}
```

1-2. クラスを使いやすくする

1. 問 1～問 4

```
package pass1;
public class HealthRecord {
    private String   name;      // 氏名
    private double   height;    // 身長 cm
    private double   weight;    // 体重 kg

    public HealthRecord(String name, double height, double weight){
        this.name   =   name;
        this.height =   height;
        this.weight =   weight;
    }

    public  double  bmi(){  // BMI 指数を計算して返す
        return 10000 * weight/ (height*height);
    }

    public String   toString(){// フィールド変数の文字列表現を返す
        return  name + "/" + height + "cm/" + weight + "kg";
    }

    public String   getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public double getHeight() {
        return height;
    }
    public void setHeight(double height) {
        this.height = height;
    }
    public double getWeight() {
        return weight;
    }
    public void setWeight(double weight) {
        this.weight = weight;
    }

}
```

アクセサメソッドを Eclipse の自動生成機能（コラム参照）で挿入。

問 5

```
package pass1;
public class Exec {
    public static void main(String[] args) {
        String[] name   = {"佐藤一郎","荒川弘子","江頭幸一"};
        double[] height = {170.2, 162.0, 175.5};
        double[] weight = {65.3, 52.5, 82.1};
        for(int i=0; i<name.length; i++){
            HealthRecord hr
                =   new HealthRecord(name[i], height[i], weight[i]);
            System.out.printf("%s / %5.2f¥n", hr.getName(), hr.bmi());
        }
    }
}
```

【解説】printf では、BMI を少数点以下 2 桁まで表示するので%5.2f という書式指定をしています。

2．問 1～問 4

```
package pass2;
public class Range {
    private double      min;
    private double      max;

    public Range(double min, double max){
        this.min    = min;
        this.max    = max;
    }

    public  boolean isOK(double a){
        return  a>=min && a<max;
    }

    public  String  toString(){
        return  "min:"+min+" - max:"+max;
    }

    public double getMin() {
        return min;
    }
    public double getMax() {
        return max;
    }
}
```

【解説】

問 3 の isOk メソッドで、return　a>=min && a<max; の部分は、

```
if(a>=min && a<max)
    return true;
else
    return false;
```

と同じ意味になります。

なぜなら、a>=min && a<max は関係式なので、true か false の値になるからです。関係式が成立していれば true、そうでなければ false の値となり、if 文で判定するのと同じ答えが返ります。

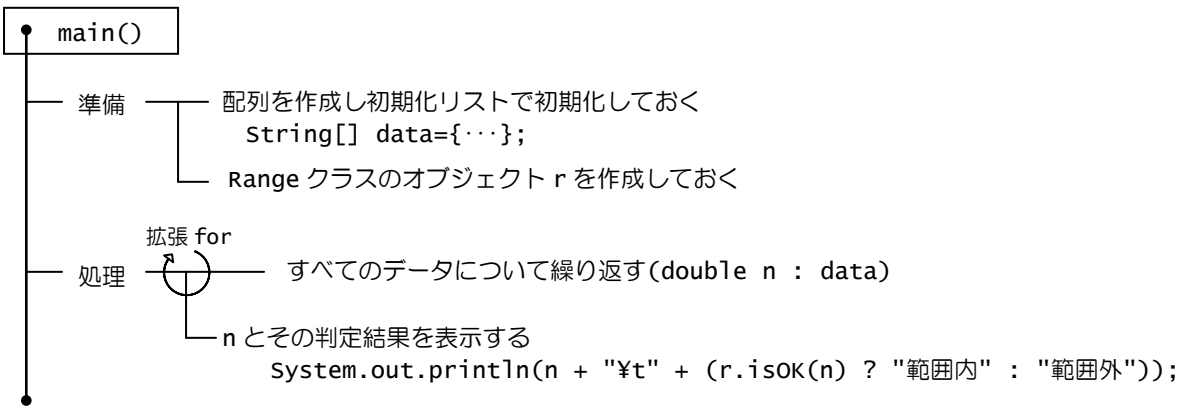
問 5

```
package pass2;
public class Exec {
    public static void main(String[] args) {
        double[] data={150.5, 75.1, 35.3, 281.2, 210.3};
        Range r = new Range(50.0,250.0);
        System.out.println(r.toString());
        for(double n : data){
            System.out.println(n + "¥t" + (r.isOK(n) ? "範囲内" : "範囲外"));
        }
    }
}
```

【解説】

最初に Range クラスのオブジェクトを作っておき、それを使って for 文の中で 5 件のデータの評価を行います。また、条件演算子?:は r.isOK(n)が true なら"範囲内"、そうでなければ"範囲外"という文字列を返すので、それを println で表示しています。if 文を使うよりも簡単になります。

次のような SPD になります。



3.

```
package pass3;
import pass1.HealthRecord;
import pass2.Range;
public class Exec {
    public static void main(String[] args) {
        String[] name
                = {"前田浩二","中村二郎","本田末男","岡村由紀","斎藤真里"};
        double[] height = {178.8, 165.7,172.1,158.8,155.6};
        double[] weight = {70.1, 72.8, 65.5, 51.3, 56.5};
        Range r         = new Range(18.5, 25.0);
        for(int i=0; i<name.length; i++){
            HealthRecord hr
                    =   new HealthRecord(name[i], height[i], weight[i]);
            double  bmi     =   hr.bmi();
            String  mark    =   (r.isOK(bmi) ? "○" : "×");
            System.out.printf("%s / %5.2f --- %s¥n", name[i], bmi, mark);
        }
    }

}
```

【解説】

1.と 2.のプログラムの組み合わせです。オブジェクト指向のプログラムではこのように複数のクラスを利用して目的の処理を行うのが普通です。

なお、String mark=(r.isOK(bmi) ? "○" : "×");を先に計算しておくのは、最後の printf 文を簡潔に書けるようにするためです。

1-3. 同じ名前で呼び出せるようにする

1.問 1～問 3

```
package pass1;
public class Player {
    private      String  name;
    private      int     level;
    private      int     hp;
    public  Player(String name, int level, int hp){
        this.name       =   name;
        this.level  =   level;
        this.hp     =   hp;
    }
    public  Player(String name){   // 問 1
        this(name, 1, 10);
    }
    public  String  toString(){
        return  name + "：level="+level+" HP="+hp;
    }
    public int damage(int p, double r){ // 問 2
        if(r>Math.random())     hp -= p;
        return  hp;
    }
    public int damage(int p){ // 問 3
        return damage(p, 0.8);
    }
}
```

【解説】

問 1　Exec クラスでは名前だけを引数に取るコンストラクタが使われているので、これがオーバーロードしなければならないコンストラクタです。つまり、name だけを引数に取るコンストラクタをオーバーロードします。

　また、実行結果をみると、level が 1、HP が 10 となっているので、オーバーロードするコンストラクタでは、level を 1、HP を 10 として this(name, 1, 10) のように引数を 3 つ取るコンストラクタを呼び出します。

問 3 は次のようにしても正解です。

```
    publicint damage(int p){
       if(0.8>Math.random())  hp -= p;
       return   hp;
    }
```

　ただ、これだと引数が二つある damage メソッドと同じような内容を記述することになりま

す。同じような内容が重複していると、将来手直しすることになった時、何か所も同じような手直しをしなければならなくなり、ミスを犯す危険性が高くなります。

そこで、次のように書くと、実際の処理を引数が二つあるメソッドに任せることができます。

```
public int damage(int p){
   int a = damage(p, 0.8);
   return   a;
}
```

ただ、これは解答のように書くと、さらに簡潔になります。damage(p, 0.8)の戻り値を、中間の変数に受けないでそのまま return 文で返す、という意味です。

```
public int damage(int p){
   return   damage(p, 0.8);
}
```

問 4

```
package pass1;
public class Exec {
    public static void main(String[] args) {
         Player   tom =   new Player ("トム", 5, 10);
         System.out.println(tom);
         tom.damage(3);
         System.out.println(tom);
    }
}
```

【解説】System.out.println(tom)は、System.out.println(tom.toString)と同じ結果になります。

2. イ カ ク

【解説】ア＝× 引数の数が同じでも型や並び順が違えばオーバーロードできる。イ＝○ 引数の個数が違えば常にオーバーロード可能。ウ＝× アクセス修飾子が違っているだけではオーバーロードにならならい。エ＝× 戻り値型が違うだけではオーバーロードにならない。オ＝× 引数の名前はオーバーロードに何の影響も及ぼさない。カ＝○ 引数の型が違えば常にオーバーロード可能。キ＝× this()はコンストラクタをオーバーロードする時だけ使える。ク＝○ オーバーロードされたメソッドは通常のメソッドと同じなので、呼び出しは自由である。

2-1. 参照はオブジェクトの分身

1.

ア	イ	ウ	エ	オ	カ
②	③	①	②	①	③

2.

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
○	×	×	×	○	○	×	○

【解説】(1)new はオブジェクトを作成してヒープ領域に置く演算です。オブジェクトをこれ以外の方法で作成することはできません。(2)参照がコピーされるだけで、オブジェクトはコピーされません。(3)配列型とクラス型では相互に代入できません。(4)コンパイルエラーになります。(5)null はオブジェクトにリンクしていない参照です。参照なので変数に代入して初期化できます。(6) 初期化したことになるので可能です。(7)null はオブジェクトにリンクしていない参照です。何かのオブジェクトを参照していることはありません。(8)数値は 0 に、文字列は null、boolean は false に初期化されます。

3. 問 1　イ

【解説】new に注目するとわかるように、a、b は 5、6 行で独立したオブジェクトとして作成しています。しかし、7 行目で a=b により a と b は同じオブジェクトを参照するようになります。そして 8 行目で b.setN(10)としているので、a.getN()でも同じ値 10 が得られます。

問 2　エ

【解説】 配列変数はキャストや自動型変換はできません。5 行目がコンパイルエラーになります。

2-2. 参照型の使用パターンを覚える

1.問 1、2、3

```
package pass1;
import mylib.ChipSet;
public class Player {
     private String name;
     private ChipSet   chips;
     public Player(String name, ChipSet chips){
         this.name  =   name;
         this.chips =   chips;
     }
     public Player(String name, int n){                    問 2
         this.name  =   name;
         chips      =   new ChipSet(n);
     }

     public int change(int p){                             問 3
         return chips.change(p);
     }

     public int getPoints(){
         return chips.getPoints();
     }
     public String getName()  { return name;}
     public String toString(){
         return name + "/" + chips.getPoints();
     }
}
```

【解説】

問 2 のように、オブジェクトを引数でうけとらない場合は、コンストラクタ内で生成してフィールド変数に代入します。

問 2 は次のように、this()を使って簡単化することができます。new ChipSet(n)は、オブジェクトを作成して返すので、そのまま（）内に書いて構いません。

```
     public Player(String name, int n){                    問 2
            this(name, new ChipSet(n));
     }
```

問 4

```
package pass1;
public class Exec {
     public static void main(String[] args) {
         Player tom =   new Player("トム", 100);
         tom.change(150);
         tom.change(-70);
         System.out.println(tom);
     }
}
```

【解説】70 点引く場合は change メソッドに-70 を指定します

2.　問 1

```
package pass2;
import pass1.Player;
public class PlayerTools {
     public static Player getWinner(Player a, Player b){
         int pa  =   a.getPoints();
         int pb  =   b.getPoints();
         if(pa>pb)      return a;
         else if(pa<pb)return b;
         else           return null;
     }
}
```

【解説】ChipSet クラスは 10 点と 1 点の 2 種類のチップを持っていますが、getPoints メソッドはその総計を返します。

```
package pass2;
import mylib.Dice;
import pass1.Player;
public class Exec {
     public static void main(String[] args) {
         Dice   dice   =   new Dice(1000);
         Player tom =   new Player("トム", dice.play());
         Player john   =   new Player("ジョン", dice.play());
         Player winner =   PlayerTools.getWinner(tom, john);
         if(winner==null)  System.out.println("引き分け");
         else           System.out.println(winner);
     }
}
```

【解説】dice.play()は 1～1000 の間の乱数を返すので、そのまま player のコンストラクタの引数に書いて構いません。

3.

```
package pass3;
import mylib.ChipSet;
public class Player {
    private String name;
    private ChipSet  chips;
    public Player(String name, ChipSet chips){
        this.name  =  name;
        this.chips =  chips;
    }
    public Player(String name, int n){
        this.name  =  name;
        chips      =  new ChipSet(n);
        /*
         * 以下のように this()を使う方法もある
         *
         *  this(name, new ChipSet(n));
         */
    }

    public int compare(Player other){
        int a   =  getPoints();
        int b   =  other.getPoints();
        if(a>b)        return 1;
        else if(a==b)  return 0;
        else           return -1;
    }

    public int change(int p){
        return chips.change(p);
    }
    public int getPoints(){
        return chips.getPoints();
    }
    public String getName()  { return name;}
    public String toString(){
        return name + "/" + chips.getPoints();
    }
}
```

【解説】

other.getPoints()は、別のオブジェクト other の総点数を返します。

問 2

```
package pass3;
import mylib.Dice;
public class Exec {
     public static void main(String[] args) {
         Dice   dice   =   new Dice(1000);
         Player tom    =   new Player("トム", dice.play());
         Player john   =   new Player("ジョン",dice.play());
         int    result =   tom.compare(john);
         if(result==1){
             System.out.println(tom);
         }else if(result==0){
             System.out.println("引き分け/"+tom.getPoints());
         }else{
             System.out.println(john);
         }
     }
}
```

【解説】

tom の compare メソッドを使って、john と比較するので、tom.compare(john)となります。tom オブジェクトの中で、tom の持ち点と john の持ち点を比較します。

4.問１

```
package pass4;

public class Stat {
    private    double[] x;
    public     Stat(double[] x){
        this.x = x;
    }
    public int        size()     { return x.length; }
    public double[]   getX()     { return x; }
    public double     get(int i) { return   x[i]; }
    public double total(){
        double sum =   0;
        for(double a : x){
            sum += a;
        }
        return sum;
    }
    public double mean() { return   total()/size(); }
    public double[]   add(double[] y){
        if(x.length!=y.length)   return null;
        double[] z =  new double[size()];
        for(int i=0; i<size(); i++){
            z[i] = x[i] + y[i];
        }
        return z;
    }
    public Stat       add(Stat other){
        if(x.length!=other.x.length)return null;
        // if(size()!=other.size()) return null; としてもよい
        double[] y = other.x;
        double[] z = add(y);
        return new Stat(z);
    }
}
```

【解説】

add メソッドでは、最初に配列のサイズが同じかどうかチェックし、異なる場合は null を返して、終了するようにしています。

下から 4 行目の double[] z = add(y);は、add(double[] y)メソッドを利用して記述を簡単にしています。こうしない場合は、for 文を書いて足し合わせる処理が必要になります。

問 2

```
package pass4;
public class Exec1 {
     public static void main(String[] args) {
         double[] a = {1.0, 2.0, 3.0, 4.0, 5.0};
         Stat   da  =   new Stat(a);
         System.out.println("合計＝" + da.total());
         System.out.println("平均＝" + da.mean());
         double[] b = {5.0, 4.0, 3.0, 2.0, 1.0};
         double[] c = da.add(b);
         if(c!=null){
             for(double s : c){
                 System.out.printf("%5.1f", s);
             }
         }else{
             System.out.println("配列は null です");
         }
     }
}
```

3-1. 継承してクラスを作る

1.問 1

```
package pass1;
public class Exec1 {
     public static void main(String[] args) {
         double[]   x   =   {1, 2, 3, 4, 5};
         Function   f   =   new Function(2, 5);
         System.out.println(f);
         for(double v  : x){
             System.out.print("("+v+", "+f.value(v)+ ") ");
         }
     }
}
```

問 2

```
package pass1;
public class Function2 extends Function {
     public Function2(double a, double b){
         super(a, b);
     }
     public double xValue(double y){
         return (y-getB())/getA();
     }
}
```

【解説】super(a, b)でスーパークラスからの継承部分を初期化する必要があります。

問 3

```
package pass1;
public class Exec2 {
     public static void main(String[] args) {
         double[]   y   =   {1, 2, 3, 4, 5};
         Function2  f   =   new Function2(2, 5);
         System.out.println(f);
         for(double v  : y){
             System.out.print("("+f.xValue(v)+", "+v+ ") ");
         }
     }
}
```

2.問 1

```
package pass2;
public class Exec1 {
     public static void main(String[] args) {
         RangeBase  r   =   new RangeBase(-1, 2);
         System.out.println(r);
     }
}
```

問 2

```
package pass2;
public class Range extends RangeBase {
     public Range(double a, double b){
         super(a, b);
     }
     public Range  shift(double x){
         double s   =   getStart() + x;
         double e   =   getEnd()   + x;
         Range r    =   new Range(s, e);
         return r;  // 上の行とまとめて return new Range(s,e); としてもよい
     }
}
```

問 3

```
package pass2;
public class Exec2 {
    public static void main(String[] args) {
         Range    r   =   new Range(-1, 2);
         Range    r1  =   r.shift(2);
         System.out.println(r + " を 2 平行移動すると " + r1);
         Range    r2  =   r.shift(-2);
         System.out.println(r + " を-2 平行移動すると " + r2);
    }
}
```

3.問 1

```
     public Item(int code, int price){
         super(code);
         this.price =   price;
     }
```

問 2　オ

【解説】Tool クラスと Item クラスは同じパッケージにあるので、public, デフォルト、protected のメンバを使うことができます。①～④はどれもこの条件に合致しています。

3-2. クラス階層をまたぐ機能を知る

1. 問 1

```
package pass1;
public class Data{
     private  double[] data;
     public Data(double[] data) {
          this.data = data;
     }
     public   double   get(int i) { return data[i]; }// i 番目の要素を返す
     public   void print(){
          for(int i=0; i<data.length; i++){
               System.out.printf("%7.2f", data[i]);
               if((i+1)%10==0) System.out.println();  // 1 行に 10 個で改行する
          }
     }
     public int size() { return data.length; }    // 要素数を返す
}
```

＜注＞ すでに配布している正誤表にある通り、問 1 では i の値のチェックをしません。

211 頁の訂正

（誤）get メソッドでは、i が正しい値（配列要素数未満）かどうかチェックしなさい

（正）get メソッドでは、i が正しい値（配列要素数未満）かどうかチェックしない

チェックしない理由は、i が不正な値であるとき、get メソッドはどのような値も返すわけにはいかないからです。このようなチェックは、Part４（318 頁）で解説している「例外を投げる」ことによって行う以外ありません。解説が後になりますので、ここでは「チェックをしない」ことにしています。

問 2

```
package pass1;
public class Stat extends Data{
     public   Stat(double[] data){
          super(data);
     }
     /* 要素の合計を返す */
     public   double   total(){
          double   t=0;
          for(int i=0; i<size(); i++) {
               t+=get(i);
          }
          return t;
     }
     /* 平均値を返す */
     public double mean() { return total()/size(); }
}
```

問 3　ア

【解説】サブクラスをスーパークラスに代入すると自動型変換がおこなわれ、エラーにならない。

問 4　ア イ

【解説】アはサブクラス型のオブジェクトを直接スーパークラス型の変数 d に代入する書き方。d は Data 型なので Data 型のメソッドしか使えない。ここで該当するのはイの print メソッドだけである。

2. 問 1

```
package pass2;
public class Player {
     private  String   name;
     private  int      score;
     public   Player(String name, int score){
          this.name  =  name;
          this.score =  score;
     }

     /* ゲームを実行する */
     public void   play()  {}  // サブクラスでオーバーライドする

     /* 相手と得点を比較し、結果を数値で返す */
     public int        compareTo(Player p) {
          return score - p.score;
     }

     /* 得点をセットする */
     public void setScore(int score) { this.score = score; }

     public int    getScore()  { return score; }
     public String  getName()  { return name;  }
     public String  toString()  { return name+"="+score; }

}
```

【解説】 score-p.score という式は、score>p.score なら正の値、score<p.score なら負の値、score==p.score なら 0 になる。メソッドは、正、負、ゼロのどれかの値を返せばよいので、この引き算の答えを return 文で返せばよい。if 文よりも簡潔な書き方。

問 2

```
package pass2;
public class DicePlayer extends Player {
     protected Dice dice;
     public DicePlayer(String name, int score){
          super(name, score);
          dice = new Dice();
     }
     @Override
     public void play(){
          int n = dice.play();
          setScore(n);
     }
}
```

【解説】アは、Dice 型のオブジェクトを new で作成してフィールド変数に代入する。イは、オーバーライドなので、@Override アノテーションを付ける。dice.play()がランダムに 1～6 の値を返すので（Dice 型のソースコードを参照）、これを setScore メソッドに渡せばよい。
単に、setScore(dice.play()); としてもよい。

問 3

```
package pass2;
public class Exec {
     public static void main(String[] args) {
          DicePlayer    p1  =    new DicePlayer("田中", 0);
          DicePlayer    p2  =    new DicePlayer("佐藤", 0);
          p1.play();
          p2.play();
          System.out.print(p1 + " "+ p2+"¥t");
          if(p1.compareTo(p2)>0){
               System.out.println(p1.getName()+"の勝ち");
          }else if(p1.compareTo(p2)<0){
               System.out.println(p2.getName()+"の勝ち");
          }else{
               System.out.println("引き分け");
          }
     }
}
```

【解説】compareTo メソッドの引数は player 型だが、このように DicePlayer 型の変数を使っても自動型変換がおこって、うまく機能する。
なお、「p1 の勝ちと表示する」には p1.getName()+"の勝ち" という文字列を表示するとよい。

3-3. オブジェクト指向の方法を会得する

1.問 1

```
package pass1;
import mylib.Student;
public class CompareStudent extends CompareSys {
     private    Student    a;
     private    Student    b;
     public     CompareStudent(Student a, Student b){
          this.a =  a;
          this.b =  b;
     }
     @Override
     protected  int        cp(){
          return a.getId() - b.getId() ;
     }
     @Override
     protected  String largeCase() {
          return a.getName()+"は"+b.getName()+"よりも後の学籍番号です" ;
     }
     @Override
     protected  String equalCase() {
          return a.getName()+"は"+b.getName()+"と同じ学籍番号です" ;
     }
     @Override
     protected  String smallCase() {
          return a.getName()+"は"+b.getName()+"よりも前の学籍番号です" ;
     }
}
```

【解説】①～④は、return 文の後に、式の形で書いてよい。式を評価した結果がメソッドの戻り値として返されます。

①では、a.getId()-b.getId()　のように引き算の式を書くと、a.getId()>b.getId()の時は正の値が返され、a.getId()==b.getId()なら 0 が、そして、a.getId()<b.getId()なら負の値が返されます。大小を if 文で調べるよりも簡単な書き方です。

問 2

```
package pass1;
public class Exec {
     public static void main(String[] args) {
          Student  a    =   new Student(100, "田中");
          Student  b    =   new Student(100, "鈴木");
          CompareSys comp =new CompareStudent(a, b);
          System.out.println(comp.compare());
     }
}
```

【解説】サブクラスである CompareStudent 型のオブジェクトを作成して、スーパークラスである CompareSys 型の変数 comp に代入します。どうしてもサブクラスの変数を使わねばなら

ない理由がない限り、より一般的なスーパークラス型の変数で受け取るのが OOP の定石です。comp.compare()では、多態性が働いて CompareStudent 型でオーバーライドしたメソッドが起動します。

2.問 1

```
package pass2;
/*
 * 2 つのオブジェクトを比較して結果を返す汎用クラス
 */
public class CompareMaster {
     private CompareUtility    comp;
     /* コンストラクタ */
     public   CompareMaster(CompareUtility comp){
          this.comp = comp;
     }
     /* オブジェクトの大小を比較した結果を表す文字列を返す */
     public   String compare(){
          int  result   =   comp.cp() ;
          return   message(result);
     }
     /*  比較結果を文字列で返す（n が正、ゼロ、負で場合分け） */
     protected String  message(int n) {
          if(n>0)       return  comp.largeCase() ;
          else if(n==0) return  comp.equalCase() ;
          else      return  comp.smallCase() ;
     }
}
```

【解説】具体的な処理は CompareUtility クラスのオブジェクト comp に委譲します。comp をフィールド変数にして、コンストラクタで受けとることにすると、クラス内のどのメソッドも comp を利用できます。

例題 4-1 では、処理を委譲するオブジェクト gu を playGame メソッドの引数として受け取っていましたが、この問題の場合は、comapre メソッドと message メソッドの両方で comp が必要なため、フィールド変数にします。委譲による場合、委譲するオブジェクトを外部から受け取ることが重要で、そのやり方はフィールド変数でもメソッドの引数でもどちらでも構いません。

問 2

```
package pass2;
import mylib.Dice;
public class CmpDiceUtility extends CompareUtility{
     private  Dice a;
     private  Dice b;
     public   CmpDiceUtility(Dice a, Dice b){
          this.a = a;
          this.b = b;
     }
     @Override
     public   int cp(){
          return   a.getN() - b.getN();
     }
     @Override
     public   String   largeCase() {
          return    "サイコロ a の目数はサイコロ b よりも大きい";
     }
     @Override
     public   String   equalCase() {
          return "サイコロ a の目数はサイコロ b と同じ";
     }
     @Override
     public   String   smallCase() {
          return "サイコロ a の目数はサイコロ b よりも小さい";
     }
}
```

問 3

```
package pass2;
import mylib.Dice;
public class Exec {
     public static void main(String[] args) {
          Dice a = new Dice(5, 6);
          Dice b = new Dice(3, 6);
          CompareUtility   c   =   new CmpDiceUtility(a,b);
          CompareMaster comp   =   new CompareMaster(c);
          System.out.println(comp.compare());
     }
}
```

【解説】最初に CmpDiceUtility 型のオブジェクトを作成して、CompareUtility 型の変数 c に代入しておきます。c はどうしても CmpDiceUtility 型の変数を使わねばならない理由がない限り、スーパークラス型の変数を使うのが OOP の定石です。

次に CompareMaster クラスのオブジェクトを作成する時、コンストラクタの引数として c を渡します。comp.compare()を実行すると、多態性が働いて CmpDiceUtility 型でオーバーライドした cp()メソッドが呼び出されます。

3-4. オブジェクト指向の方法を実践する

1．問 1

```
package pass1;
import mylib.Input;
public class LocalLogin extends LoginSys{
     /* id と pw の組みは、admin/bnj123、tom/upn111、john/kpio22  です */
     private static final  String[] id = {"admin","tom","john"};
     private static final  String[] PW = {"bnj123","upn111","kpio22"};
     /* 入力したパラメータを記憶しておくためのフィールド変数 */
     private  String  userid;     // ユーザー名
     private  String  password;   // パスワード
     @Override
     protected void receiveParam() {
          userid   =   Input.getString("ユーザー名");
          password =   Input.getString("パスワード");
     }
     @Override
     protected int auth() {
          for(int i=0; i<id.length; i++){
               if(ID[i].equals(userid)){
                    if(PW[i].equals(password))  return  SUCCESS;
                    else                        return  ERROR;
               }
          }
          return   ERROR;
     }
}
```

【解説】LoginSys の SUCCESS、ERROR や、LocalLogin の配列 ID や PW など、定数として使いたい変数は、一般に static final とします。static を付けるとオブジェクト毎ではなくクラスにただひとつだけの静的な変数になります。また、final は一度初期値を代入したら、それ以降、値を変更できなくします。この結果、変数を定数として使うことができます。定数として使う変数は、区別するために大文字を使うのが慣例です。

さて、auth メソッドでは、最初に userid が配列 id の中にあるかどうか調べ、あった時だけパスワードを照合します。userid が id[i]に等しければ、password が pw[i]に等しいかどうかをチェックします。

問 2

```
package pass1;
public class Exec {
     public static void main(String[] args) {
          LoginSys ls  =   new  LocalLogin();
          int result = ls.login();
          if(result==LoginSys.SUCCESS) System.out.println("ログイン成功");
          else                         System.out.println("ログイン失敗");
     }
}
```

【解説】LoginSys ls = new LocalLogin();のように、オブジェクトを作成したら、スーパークラスの変数に代入します。login メソッドを確かめるので、より一般的なスーパークラスの変数を使います。login メソッドの中では、多態性が働いて、サブクラスでオーバーロードしたメソッドが起動します。

＜別解＞

```
package pass1;
import static pass1.LoginSys.SUCCESS;
public class Exec_B {
     public static void main(String[] args) {
          LoginSys ls  =  new  LocalLogin();
          int result = ls.login();
          if(result==SUCCESS)   System.out.println("ログイン成功");
          else                  System.out.println("ログイン失敗");
     }
}
```

【解説】

これは、static import という書き方を使った例です。static import とは、static 変数や static メソッドを、クラス名を指定せずに使用する機能です。これにより、使用するたびにクラス名を記述する必要がなくなります。

それには、キーワードとして import static を使用し、インポートしたい static 変数や static メソッドを指定します。 例えば次のように指定すると、クラス名を付けないでクラス変数 SUCCESS を使うことができます。

```
import static pass1.LoginSys.SUCCESS;
```

また、次のように書くと、pass1 パッケージの LoginSys クラスのすべてのクラス変数をクラス名なしで使うことができます。

```
import static pass1.LoginSys.*;
```

まとめると、static インポートとは、static 変数や static メソッドをクラス名を指定せずに使用する機能です。これにより、使用するたびにクラス名を記述する必要がなくなります。キーワードとして import static を使用し、インポートしたい static 変数や static メソッドを指定します。

【構文】

import static パッケージ名.クラス名.static 変数名;

import static パッケージ名.クラス名.static メソッド名;

import static パッケージ名.クラス名.*;

2．問 1

```
package pass2;
public abstract class AbsProduct extends Product {
     public        AbsProduct(String code, String name){
          super(code, name);
     }
     public abstract int price();
}
```

【解説】price メソッドを拡張しただけでも、コンストラクを作成し、super コンストラクタを呼び出す必要があります。抽象メソッドを持つので抽象クラスになります。abstract キーワードが必要な点に注意してください。

問 2

```
package pass2;
public interface Shipping {
     String info();     // 製品の情報（品名、コード、価格）を文字列にして返す
     int shipping();    // 製品の送料を返す
     int handling();    // 製品の手数料を返す
}
```

問 3

```
package pass2;
public class Handbag extends AbsProduct implements Shipping {
     private  int size;
     public Handbag(String code, String name, int size) {
          super(code, name);
          this.size=    size;
     }
     @Override
     public int price() {
          String    type = code.substring(0,3);
          if(type.equals("SIG"))       return  10000;
          else if(type.equals("SHO"))  return  12000;
          else                         return  15000;
     }
     @Override
     public int shipping() {
          if(size==1)           return  500;
          else if(size==2)      return  700;
          else                  return  1000;
     }
     @Override
     public int handling() {
          if(size>=2)   return  200;
          else          return  0;
     }
     @Override
     public String info() {
          return name+"¥t"+ code + "¥t" + price();
     }
}
```

【解説】extends と implements キーワードを使って、継承とインタフェースの実装を宣言します。コンストラクタでは super コンストラクタで code と name をスーパークラスに渡します。size フィールドへの代入は super コンストラクタの後に書かねばなりません。

AbsProduct クラスから継承した price メソッドをオーバーロードし、Shipping インタフェースが規定している shipping メソッド、handling メソッド、info メソッドを実装します。どれも@Override アノテーションが付いていますが、インタフェースでは、親クラスにオーバーライドされるメソッドがあるわけではないので、オーバーライドではなく、単に作成したメソッドが増えるだけです。そのため、これを実装と呼ぶことに注意してください。

substring メソッドは、String クラスのオブジェクトが持つメソッドです。String クラスには数多くのメソッドがありますが、substring は文字列の一部分をコピーした部分文字列を返します。文字列の先頭から 0 番目、1 番目、2 番目、･･･と数えます。先頭から 3 文字の部分文字列であれば、コピーする範囲として 0～3 番目を指定します。これは、0 番目から 3 番目の直前（＝2 番目）までコピーするという指定です。少し癖のある指定方法なので注意してください。

問 4

```
package pass2;
public class Chair extends AbsProduct implements Shipping {

    public Chair(String code, String name) {
        super(code, name);
    }
    @Override
    public int price() {
        char c = code.charAt(0);
        if(c=='S')          return 12000;
        else if(c=='O')     return 20000;
        else                return 30000;
    }
    @Override
    public int shipping() {
        char c = code.charAt(0);
        if(c=='S')          return 1200;
        else if(c=='O')     return 1400;
        else                return 2000;
    }
    @Override
    public int handling() {
        return 200;
    }
    @Override
    public String info() {
        return name+"¥t"+ code + "¥t" + price();
    }
}
```

【解説】charAt メソッドは String クラスのオブジェクトが持つメソッドです。文字列の指定した位置にある 1 文字を取り出します。位置指定は、substring メソッドと同じく、文字列の先頭から 0 番目、1 番目、2 番目、･･･と数えます。ここでは先頭の文字なので、code.charAt(0); です。

問 5

```
package pass2;
public class Dumbbell extends AbsProduct implements Shipping {

     int weight;
     public Dumbbell(String code, String name, int weight) {
          super(code, name);
          this.weight   =   weight;
     }
     @Override
     public int price() {
          String    type =  code.substring(0,3);
          if(type.equals("SQR"))           return  1000;
          else if(type.equals("RND"))      return  1100;
          else                             return  900;
     }
     @Override
     public int shipping() {
          if(weight<2)          return  300;
          else if(weight<5)     return  400;
          else                  return  500;
     }
     @Override
     public int handling() {
          return 100;
     }
     @Override
     public String info() {
          return name+"¥t"+ code + "¥t" + price();
     }
}
```

問 6

```
package pass2;
public class Exec {
     public static void main(String[] args) {
          Shipping[] sp  = new Shipping[3];
          sp[0] =  new  Handbag("SIG101", "クリスティン", 1);
          sp[1] =  new  Chair("S203", "籐スツール" );
          sp[2] =  new  Dumbbell("RND11", "丸型ダンベル", 3);
          System.out.println("品　名¥t¥tCODE¥t 価格¥t 送料¥t 手数料");
          for(Shipping s : sp){
               System.out.println(s.info()+
                              "¥t"+s.shipping()+"¥t"+s.handling());
          }
     }
}
```

【解説】Handbag、Chair、Dumbbell 型の各オブジェクトはどれも Shipping インタフェースを実装しているので、Shipping 型の変数に代入できます。ここでは、Shipping 型の配列変数 sp を作成し、その要素にひとつづつ代入しています。単一の変数に代入するのも、配列の

要素に代入するのも、実質的には同じ操作です。

拡張 for 文を使って、sp からひとつずつ Shipping 型の変数 s にオブジェクトを取り出します。Shipping 型の変数は、info、shipping、handling の各メソッドを使用できます。これらを使って解答のように出力します。

Shipping インタフェース型の変数を使うことで、多態性により、これらのメソッドはクラスごと定義された異なる処理を行っていることに注意してください。

4-1. 例外を使いこなす

1.

①Throwable　②RuntimeException　③実行時例外
④try-catch 文　⑤スーパークラス　⑥サブクラス

2.

```
package pass2;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Exec {
    public static void main(String[] args){
        try(BufferedReader in =
                new BufferedReader(new FileReader("src¥¥mylib¥¥Dice.java")); ) {
            String s;
            int line=0, ch=0;    // 行数と文字数を 0 にしておく
            while((s= in.readLine()) != null){
                line++;              // 行数を 1 増やす
                ch += s.length();    // 文字数を加える
            }
            System.out.println("行数="+line);
            System.out.println("字数="+ch);
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

【解説】

BufferedReader の readLine()メソッドは 1 行分のデータを読み込みます。テキストファイルは目に見える文字だけからなるファイルですが、1 行の終わりを示すために行末にあたる部分には改行コードがあります。1 行分とはこの改行コードの直前までのデータです。

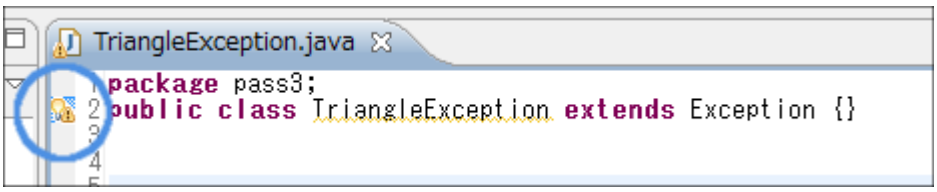
3.問 1

```
package pass3;
public class TriangleException extends Exception {}
```

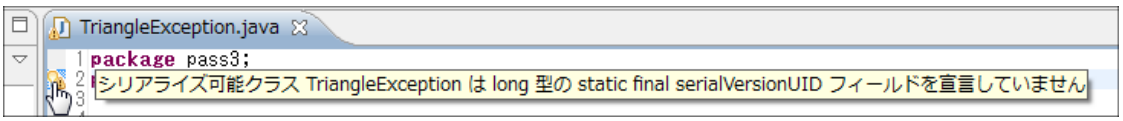
【解説】

TriangleException クラスは、Exception クラスを継承しただけのクラスです。

ただ、Eclipse では次のような警告マークが表示されます。



警告内容は次のようなものです。

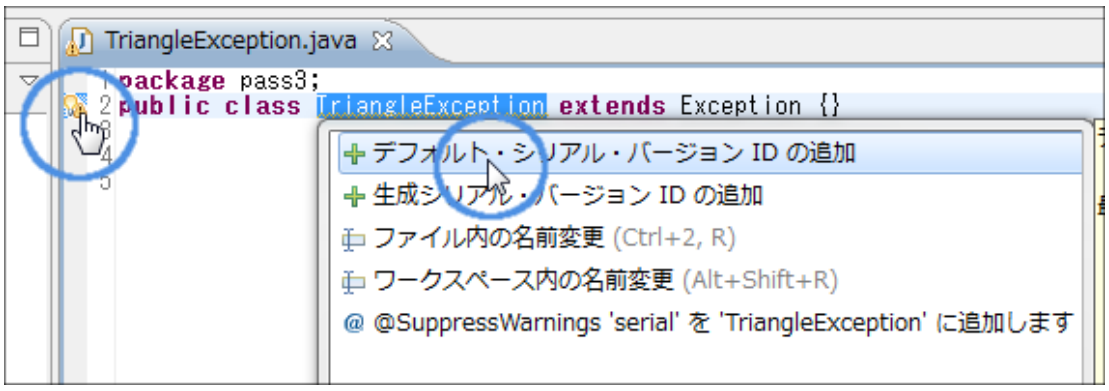


これは、オブジェクト全体をそのままファイルなどに保存できる形式に変換する「シリアライズ」という機能に関連した警告です。シリアライズ変換したオブジェクトはファイルに保存したり、通信で送ったりできます。

シリアライズ変換されたオブジェクトデータは、逆変換してもとのオブジェクトに復元できますが、プログラムの改訂によりオブジェクトの定義が変更される場合があります。そのため、変換して保存していたものでも、新しいプログラムでは逆変換できないことがあります。

そこで普通はバージョン番号をオブジェクトの定数フィールドとして作っておきます。警告は、その定数フィールドがない、と言っているわけです。

シリアライズしないのであれば無視してもよい警告ですが、次のように、警告マークをクリックし、ポップアップメニューで［デフォルトシリアルバージョン ID の追加］を選択すると定数フィールドを自動作成してくれます。



ID を追加したクラスは次のようになりますが、解答としてはどちらでも構いません。

```
package pass3;
public class TriangleException extends Exception {
     private static final long serialVersionUID = 1L;
}
```

問 2

```
package pass3;
public class Triangle {
     private  double   a;
     private  double   b;
     private  double   c;
     public Triangle(double a,double b,double c) throws TriangleException {
          if(a>=b+c || b>=c+a || c>=a+b){
               throw    new TriangleException();
          }
          this.a    =    a;
          this.b    =    b;
          this.c    =    c;
     }
     public    double   area(){
          double    s    =   (a+b+c)/2;
          return    Math.sqrt(s*(s-a)*(s-b)*(s-c));
     }
}
```

【解説】

コンストラクタでは、if で三角形の成立条件を調べ、不成立なら TriangleException を投げます。

問 3

```
package pass3;
import mylib.Input;
public class Exec {
     public static void main(String[] args) {
          Triangle t    =   null;
          while(t==null){
               double   a   =    Input.getDouble("辺 A");
               double   b   =    Input.getDouble("辺 B");
               double   c   =    Input.getDouble("辺 C");
               try {
                    t = new Triangle(a, b, c);
               } catch (TriangleException e) {
                    System.out.println("辺の長さが正しくない¥n");
               }
          }
          System.out.println("面積="+t.area());
     }
}
```

【解説】

5 行目で Triangel t = null;としているところがポイントです。null はオブジェクトにリンクしていない参照です。オブジェクトが null で初期化できることは Part2-1 でも解説しました。

これにより、while 文の中で、オブジェクトが null である間、オブジェクトの作成処理を繰り返すことができます。

問 4

```
package pass3;
import java.io.IOException;
import java.io.PrintWriter;
import mylib.Input;
public class Exec2 {
     public static void main(String[] args) {
          Triangle t    =   null;
          while(t==null){
               double   a   =   Input.getDouble("辺 A");
               double   b   =   Input.getDouble("辺 B");
               double   c   =   Input.getDouble("辺 C");
               try {
                    t    =   new Triangle(a, b, c);
               } catch (TriangleException e) {
                    System.out.println("辺の長さが正しくない¥n");
               }
          }
          try(PrintWriter out = new PrintWriter("area.txt");){
               out.println("面積="+t.area());
          }catch(IOException e){
               e.printStackTrace();
          }
     }
}
```

【解説】

青枠内が書き変えた部分です。リソース付き try 文なので close()メソッドを実行する必要はありません。作成された area.txt を見るためには Eclipse でプロジェクトをクリックして F5 キーをタイプしてください。あるいは、プロジェクトをマウスの右ボタンでクリックして［リフレッシュ］を選択します。

4-2. データ構造の効用を知る

1. 問 1

```
package pass1;
import mylib.Book;
public class Bookshelf {
     private  Book[]   books;
     public   Bookshelf(Book[] books){
         this.books = books;
     }
     public int size(){
         return books.length;
     }
     public Book get(int i){
         return books[i];
     }
}
```

問 2

```
package pass1;
import mylib.Book;
public class Exec {
     public static void main(String[] args) {
         Book[] books = { new Book("吾輩は猫である","夏目漱石"),
                          new Book("杜子春","芥川龍之介"),
                          new Book("雪国","川端康成") };
         Bookshelf  bs =   new Bookshelf(books);
         for(int i=0; i<bs.size(); i++){
             System.out.print("["+(i+1)+"]");
             System.out.println(bs.get(i));
         }
     }
}
```

【解説】for 文の代わりに拡張 for 文を使うと次のようになる

```
int k=1;
for(Book bk : books){
    System.out.println("["+(k++)+"]"+ bk);
}
```

2．問１

```
package pass2;
public class Measurement {
     private String name;
     private double height;
     private double weight;
     public Measurement(String name, double height, double weight){
         this.name     =   name;
         this.height   =   height;
         this.weight   =   weight;
     }
     public String getName()     { return   name; }
     public double getHeight()   { return   height; }
     public double getWeight()   { return   weight; }
     public String toString(){
         return name + " "+ height + "cm/ "+weight+"kg";
     }
}
```

問 2

```
package pass2;
import java.util.ArrayList;
public class MeasurementList {
     private ArrayList<Measurement> ls;
     public  MeasurementList(){
         ls = new ArrayList<Measurement>();
     }
     public void   add(Measurement m)    {ls.add(m);}
     public int    size()                {returnls.size(); }
     public Measurement get(int i)       {returnls.get(i); }
     public Measurement average(){
         double     heightAve=0, weightAve=0;
         for(Measurement me : ls){
             heightAve += me.getHeight();
             weightAve += me.getWeight();
         }
         /**  measurement 型オブジェクトを作成して戻り値として返す **/
         Measurement m =
            new Measurement("平均値", heightAve/ls.size(), weightAve/ls.size());
         return m;
     }
     public void print(){
         for(Measurement me : ls){
             System.out.println(me);
         }
     }
}
```

問 3

```
package pass2;
public class Exec {
     public static void main(String[] args) {
         MeasurementList ml    =   new MeasurementList();
         ml.add(new Measurement("大木  太郎", 168.5, 68.1));
         ml.add(new Measurement("中村  二郎", 172.4, 70.5));
         ml.add(new Measurement("古田  三郎", 180.3, 75.0));
         ml.add(new Measurement("池田  志郎", 182.3, 75.5));
         ml.add(new Measurement("川中  吾朗", 178.5, 81.2));
         Measurement ave = ml.average();
         ml.print();
         System.out.println();
         System.out.println(ave); // オブジェクトを出力→toString メソッドが起動
     }
}
```

【解説】平均値の出力は、平均値の入っている ave をそのまま出力します。
　　これは、ave.toString()を出力するのと同じです。

3.

```
package pass3;
import java.util.ArrayList;
import mylib.Card;
public class Deck {
     private ArrayList<Card> deck;
     public Deck(){
         deck = new ArrayList<>();
         init();
     }
     public void   init(){
         deck.clear();
         for(int i=1; i<=52; i++){
             deck.add(new Card(i));
         }
     }
     public int size(){
         return deck.size();
     }
     public Card get(int i){
         return deck.get(i);
     }
     public Card draw(){
         int p = (int)(Math.random()*deck.size());
         return deck.remove(p);
     }
     public void print(){
         for(int i=0; i<deck.size(); i++){
             System.out.println(deck.get(i));
         }
     }
}
```

【解説】draw は、deck の要素の中で何番目を削除するか乱数で決定します。削除する番号ですから、 0 から deck.seize-1 の間の数です。(int)(Math.random()*deck.size())としているのはそのためです。

　また、deck.remove(p)では、deck の p 番目の要素を削除し、その要素を返します。これをそのまま return で戻り値として返しています。

問 2

```
package pass3;
public class Exec {
     public static void main(String[] args) {
         Deck    d   =   new Deck();
         for(int i=0; i<5; i++){
             System.out.println(d.draw());
         }
         System.out.println("残り"+d.size()+"枚");
     }
}
```

問 3

```
package pass3;
import mylib.Card;
import sample5.Hand;
public class Exec2 {
     public static void main(String[] args) {
         Deck    deck    =   new Deck();
         Hand    hand    =   new Hand();
         for(int i=0; i<5; i++){
             Card    c   =   deck.draw();   // deck から 1 枚取り出す
             hand.add(c);                   // それを Hand に追加する
         }
         // Hand の要素をすべて表示する
         for(int i=0; i<5; i++){
             System.out.println(hand.get(i));
         }
     }
}
```

4-3. マルチスレッドで平行処理を行う

1.

①	②	③	④	⑤	⑥	⑦	⑧
カ	キ	サ	ウ	コ	シ	エ	オ

2.

問 1

```
package pass2;
import mylib.DummyTask;
public class MyUploader implements Runnable{
     @Override
     public void run() {
          for(int i=0; i<5; i++){
               DummyTask.oneSecond();
               System.out.println("uploading");
          }
     }
}
```

問 2

```
package pass2;
public class Exec1 {
     public static void main(String[] args) {
          Thread   t   =   new Thread( new MyUploader());
          t.start();
     }
}
```

問 3

```
package pass2;
import java.util.concurrent.*;
public class Exec2 {
     public static void main(String[] args) {
          ExecutorService es = Executors.newSingleThreadExecutor();
          es.execute(new MyUploader());
          es.execute(new MyUploader());
          es.shutdown();
     }
}
```

問 4

```
package pass2;
import java.util.concurrent.*;
public class Exec3 {
     public static void main(String[] args) {
          ExecutorService es = Executors.newCachedThreadPool();
          for(int i=0; i<50; i++){
               es.execute(new MyUploader());
          }
          es.shutdown();
     }
}
```

問 5　Exec3 が速く終了する

【解説】Exec3 は、タスクの実行回数は多いがスレッドプール内でたくさんのスレッドを使って同時並行的に行うので、およそ 5 秒ですべての処理が終わる。一方、Exec2 は Single タイプなので、1 つのスレッドでタスクを 2 回実行することになり、約 10 秒かかる。

4-4. スレッド間での値の受け渡しを理解する

1.

問 1

```
package pass1;
public class Exec1 {
     public static void main(String[] args) {
          Friend   friend =new Friend();
          System.out.println(friend);
     }

}
```

問 2

```
package pass1;
import java.util.concurrent.*;
import mylib.DummyTask;
public class Sender implements Callable<Integer> {
     private Friend friend;
     public Sender(Friend friend) {
          this.friend = friend;
     }
     @Override
     public Integer call() throws Exception {
          if (friend.getMail() == null || friend.getMail().length() < 3) {
               throw new Exception();
          }
          System.out.println(friend);
          DummyTask.nSeconds(1.5);
          return 1;
     }
}
```

問 3

```
package pass1;
import java.util.concurrent.*;
public class Exec2 {
    public static void main(String[] args) {
         Friend            friend  =   new Friend();
         ExecutorService es        =   Executors.newSingleThreadExecutor();
         Future<Integer>   f       =   es.submit(new Sender(friend));
         try {
              int  n = f.get(3, TimeUnit.SECONDS);
              System.out.println(n+"件送信しました");
         } catch (InterruptedException | ExecutionException | TimeoutException e) {
              System.err.println(e+"が発生しました");
              f.cancel(true);
         }
         es.shutdown();
    }
}
```

問 4

```
package pass1;
import java.util.ArrayList;
import java.util.concurrent.*;
public class SendMails {
    private  ArrayList<Friend> list;
    public   SendMails(ArrayList<Friend> list){
         this.list=   list;
    }
    public   int  send(){
         ArrayList<Future<Integer>> flist = new ArrayList<>();
         ExecutorService es = Executors.newCachedThreadPool();
         for(int i=0; i<list.size(); i++){
              Friend f=list.get(i);
              Future<Integer> ft=es.submit(new Sender(f));
              flist.add(ft);
              /*
               * 上記の３行をまとめて次のように書いてもよい
                 flist.add(es.submit(new Sender(list.get(i))));
               */
         }
         int n=0;
         for(int i=0; i<list.size(); i++){
              Future<Integer>   f   =   flist.get(i);
              try {
                   n+=f.get(3, TimeUnit.SECONDS);
              } catch(InterruptedException | ExecutionException | TimeoutException e) {
                   System.err.println(e+"が発生しました");
                   f.cancel(true);
              }
         }
         es.shutdown();
         return   n;
    }
}
```

問 5

```
package pass1;
import java.util.ArrayList;
public class Exec3 {
    public static void main(String[] args) {
         ArrayList<Friend> list    =   new ArrayList<>();
         for(int i=0; i<5; i++){
             list.add(new Friend());
         }
         SendMails s = new SendMails(list);
         int  n = s.send();
         System.out.println(n + "件のメールを送信しました");
    }
}
```