

# Java プログラミング基礎

例題解答例

## 例題 1 変数、リテラル、演算子

- 変数、定数の宣言、計算等を行う

実行結果

```
aとbのビット論理積は8<
a+=2は12<
1000円にかかる税額は80.0<
```

名前 Reidai1

```
1. public class Reidai1 {
2.     public static void main(String[] args) {
3.         int answer;
4.         int a = 10;        // 2進数では 0b1010
5.         int b = 13;        // 2進数では 0b1101
6.         answer = a & b;
7.         System.out.println("a と b のビット論理積は"+answer);
8.
9.         a+=2;
10.        System.out.println("a+=2 は"+a);
11.
12.        final float tax = 0.08F;
13.        int money = 1000;
14.        System.out.println("1000 円にかかる税額は"+money*tax);
15.
16.    }
17. }
```

## 例題 2 配列

- int 型配列と String 型配列を用意し、値の格納と表示を確認する

実行結果

```
配列 su[0] は 10 ←  
配列 str[0] は A ←  
配列 su[1] は 20 ←  
配列 str[1] は B ←  
配列 su[2] は 30 ←  
配列 str[2] は C ←  
配列 su[3] は 40 ←  
配列 su[4] は 50 ←
```

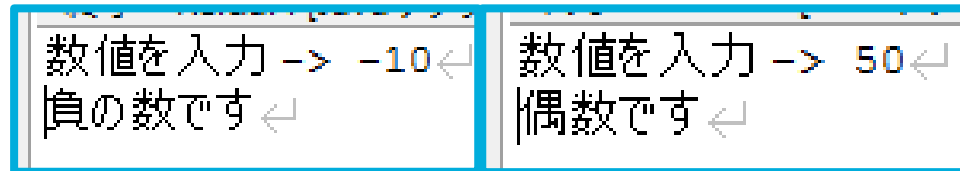
名前 Reidai2

```
1. public class Reidai2 {  
2.     public static void main(String[] args) {  
3.         int i = 0;  
4.         int[] su = new int[5];  
5.         String[] str = {"A","B","C"};  
6.         su[0] = 10;  
7.         su[1] = 20;  
8.         su[2] = 30;  
9.         su[3] = 40;  
10.        su[4] = 50;  
11.  
12.        System.out.println("配列 su["+ i + "]は"+su[i]);  
13.        System.out.println("配列 str["+ i + "]は"+str[i]);  
14.        i++;  
15.        System.out.println("配列 su["+ i + "]は"+su[i]);  
16.        System.out.println("配列 str["+ i + "]は"+str[i]);  
17.        i++;  
18.        System.out.println("配列 su["+ i + "]は"+su[i]);  
19.        System.out.println("配列 str["+ i + "]は"+str[i]);  
20.        i++;  
21.        System.out.println("配列 su["+ i + "]は"+su[i]);  
22.        i++;  
23.        System.out.println("配列 su["+ i + "]は"+su[i]);  
24.        i++;  
25.    }  
26. }
```

### 例題 3 if-2 分岐処理

- キーボードから入力した値が負の数であれば「負の数です」と表示してプログラムを終了する
- さらに、正の数で偶数なら「偶数です」、奇数ならば「奇数です」と表示する
- 今回の例では、0 は正の数、偶数と同じ振る舞いとして処理しています

実行結果



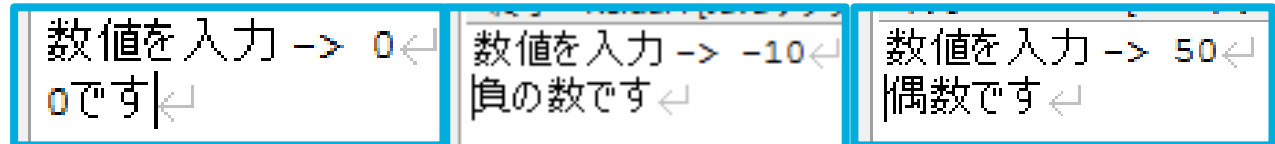
名前 Reidai3

```
1. import java.util.Scanner;
2.
3. public class Reidai3 {
4.
5.     public static void main(String[] args) {
6.         System.out.print("数値を入力 -> ");
7.         Scanner scan = new Scanner(System.in);
8.         int input = scan.nextInt();
9.
10.        if(input < 0 ){
11.            System.out.println("負の数です");
12.            System.exit(0);           // プログラムの終了
13.        }
14.
15.        if( input % 2 == 0 ){
16.            System.out.println("偶数です");
17.        }else{
18.            System.out.println("奇数です");
19.        }
20.    }
21. }
```

#### 例題 4 if-多分岐処理

- キーボードから入力した値が0であれば「0です」、負の数であれば「負の数です」と表示してプログラムを終了する
- さらに、正の数で偶数なら「偶数です」、奇数ならば「奇数です」と表示する

実行結果



名前 Reidai4

```
1. import java.util.Scanner;
2.
3. public class Reidai4 {
4.
5.     public static void main(String[] args) {
6.         System.out.print("数値を入力 -> ");
7.         Scanner scan = new Scanner(System.in);
8.         int input = scan.nextInt();
9.
10.        if(input == 0){
11.            System.out.println("0です");
12.            System.exit(0);          // プログラムの終了
13.        }else if(input < 0 ){
14.            System.out.println("負の数です");
15.            System.exit(0);          // プログラムの終了
16.        }else if( input % 2 == 0 ){
17.            System.out.println("偶数です");
18.        }else{
19.            System.out.println("奇数です");
20.        }
21.    }
22. }
```

### 例題5 switch 文

- キーボードから入力した値が 0、または負の数であれば「正の数を入力してください」と表示してプログラムを終了する
- さらに、正の数で偶数なら「偶数です」、奇数ならば「奇数です」と表示する

実行結果

```
数値を入力 -> -1↵  
正の数を入力してください↵
```

名前 Reidai5

```
1. import java.util.Scanner;  
2.  
3. public class Reidai5 {  
4.  
5.     public static void main(String[] args) {  
6.         System.out.print("数値を入力 -> ");  
7.         Scanner scan = new Scanner(System.in);  
8.         int input = scan.nextInt();  
9.  
10.        if(input == 0 || input < 0 ){  
11.            System.out.println("正の数を入力してください");  
12.            System.exit(0);           // プログラムの終了  
13.        }  
14.  
15.        switch(input%2){  
16.            case 0:  
17.                System.out.println("偶数です");  
18.                break;  
19.            case 1:  
20.                System.out.println("奇数です");  
21.                break;  
22.        }  
23.    }  
24. }
```

## 例題 6 繰り返し文

- 掛け算九九のプログラムを作成する

実行結果

```
while文による掛け算九九表示↵
1の段： 1  2  3  4  5  6  7  8  9  ↵
2の段： 2  4  6  8 10 12 14 16 18  ↵
3の段： 3  6  9 12 15 18 21 24 27  ↵
4の段： 4  8 12 16 20 24 28 32 36  ↵
5の段： 5 10 15 20 25 30 35 40 45  ↵
6の段： 6 12 18 24 30 36 42 48 54  ↵
7の段： 7 14 21 28 35 42 49 56 63  ↵
8の段： 8 16 24 32 40 48 56 64 72  ↵
9の段： 9 18 27 36 45 54 63 72 81  ↵
```

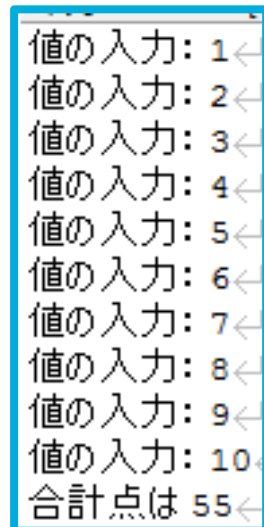
名前 Reidai6

```
1. public class Reidai6 {
2.
3.     public static void main(String[] args) {
4.         int i = 1,j = 1;
5.         System.out.println("while 文による掛け算九九表示");
6.         while( i < 10){
7.             System.out.print(" " + i + "の段 : " );
8.             while(j < 10){
9.                 System.out.print(" " + i*j + " ");
10.                j++;
11.            }
12.            System.out.println();
13.            j=1;
14.            i++;
15.        }
16.        System.out.println("for 文による掛け算九九表示");
17.        for(i = 1 ; i < 10 ; i++){
18.            System.out.print(" " + i + "の段 : " );
19.            for(j = 1 ; j < 10 ; j++){
20.                System.out.print(" " + i*j + " ");
21.            }
22.            System.out.println();
23.        }
24.    }
25. }
```

## 例題 7 繰り返し文

- キーボードから値を 10 個入力し、int 型配列 su に格納する
- その後配列に格納された値の合計を求める

実行結果



```
値の入力: 1<
値の入力: 2<
値の入力: 3<
値の入力: 4<
値の入力: 5<
値の入力: 6<
値の入力: 7<
値の入力: 8<
値の入力: 9<
値の入力: 10<
合計点は 55<
```

名前 Reidai7

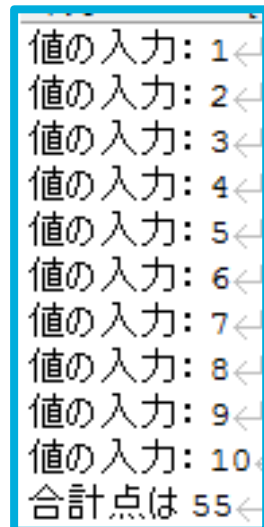
```
1. import java.util.Scanner;
2.
3. public class Reidai7 {
4.
5.     public static void main(String[] args) {
6.         int[] su = new int[10];
7.         int cnt = 0, answer = 0;
8.         Scanner scan = new Scanner(System.in);
9.
10.        do{
11.            System.out.print("値の入力: ");
12.            su[cnt] = scan.nextInt();
13.            cnt++;
14.        }while(cnt < 10);
15.        for(int i = 0 ; i < su.length ; i++){
16.            answer += su[i];
17.        }
18.        System.out.println("合計点は " + answer);
19.    }
20. }
```



## 例題 8 繰り返し文

- キーボードから値を 10 個入力し、int 型配列 su に格納する
- その後配列に格納された値の合計を求める

実行結果



```
値の入力: 1<
値の入力: 2<
値の入力: 3<
値の入力: 4<
値の入力: 5<
値の入力: 6<
値の入力: 7<
値の入力: 8<
値の入力: 9<
値の入力: 10<
合計点は 55<
```

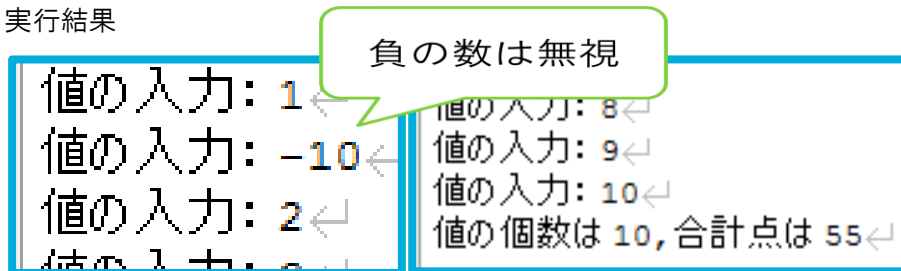
名前 Reidai8

```
1. import java.util.Scanner;
2.
3. public class Reidai8 {
4.
5.     public static void main(String[] args) {
6.         int[] su = new int[10];
7.         int cnt = 0, answer = 0;
8.         Scanner scan = new Scanner(System.in);
9.
10.        do{
11.            System.out.print("値の入力: ");
12.            su[cnt] = scan.nextInt();
13.            cnt++;
14.        }while(cnt < 10);
15.        for(int i : su){
16.            answer += i;
17.        }
18.        System.out.println("合計点は " + answer);
19.    }
20. }
```

## 例題9 break,continue

- キーボードから値を10個入力し、int型配列suに格納する
- ただし、
  - 入力した値が0ならばキー入力は終了する
  - 入力した値が負の数であれば、配列suには格納しないでキーボード入力を継続する
  - 結果として、0が途中で入力されると値が10個分入らない可能性がある。また負の数が入力されると無視されるので、正の数が10個入るまでキーボード入力が繰り返される
- 配列に格納された値の合計を表示する

実行結果



値の入力: 1  
値の入力: -10  
値の入力: 2  
値の入力: 8  
値の入力: 9  
値の入力: 10  
値の個数は 10, 合計点は 55

負の数は無視

名前 Reidai9

```
1. import java.util.Scanner;
2.
3. public class Reidai9 {
4.
5.     public static void main(String[] args) {
6.         int[] su = new int[10];
7.         int cnt = 0, answer = 0;
8.         Scanner scan = new Scanner(System.in);
9.
10.        do{
11.            System.out.print("値の入力: ");
12.            su[cnt] = scan.nextInt();
13.            if(su[cnt] == 0){
14.                break;
15.            }else if(su[cnt] < 0){
16.                continue;
17.            }
18.            answer += su[cnt];
19.            cnt++;
20.        }while(cnt < 10);
21.        System.out.println("値の個数は " + cnt + ", 合計点は " + answer);
22.    }
23. }
```

## 例題 10 クラスの作成

- Sensor クラスの定義をする

データ型	名前	説明	設定値（例）
String	productName	製品名	POLY35
String	manufacture	製造者名	ポリテケ
int	value	値	0

名前	説明	戻り値	引数
get	センサの値を取得	int	なし

名前 Sensor

```
1. public class Sensor {
2.     String productName = "POLY35";
3.     String manufacture = "ポリテケ";
4.     int value;
5.
6.     public int get() {
7.         return value;
8.     }
9. }
```

## 例題 11 クラスのインスタンス化とメンバの利用

- 例題 10 で作成したクラス Sensor のインスタンスを生成して利用する

実行結果

製造者名: ポリテケ

製品名: POLY35

取得値: 0

名前 TestSensor

```
1. public class TestSensor {
2.     public static void main(String[] args) {
3.         Sensor sensor = new Sensor();
4.
5.         System.out.println("製造者名: " + sensor.manufacture);
6.         System.out.println("製品名: " + sensor.productName);
7.
8.         int value = sensor.get();
9.         System.out.println("取得値: " + value);
10.    }
11. }
```

## 例題 12 コンストラクタ

- 例題 10 で作成した `Sensor` クラスを利用して暗黙のコンストラクタの確認をする

例題 11 とソースコードは同じ

### 例題 13 コンストラクタ

- 例題 10 で作成した Sensor クラスに以下のコンストラクタを追加し、確認する
- 引数を 2 つ取るコンストラクタ

引数	振る舞い
String productName	引数 productName の値をフィールド product に代入する

- 引数を 1 つ取るコンストラクタ

引数	振る舞い
String productName	引数 productName の値をフィールド product に代入する
String manufacture	引数 manufacture の値をフィールド manufacture に代入する

実行結果

No.1

製造者名: ポリテケ

製品名: POLY35

値: 0

No.2

製造者名: ポリテケ

製品名: POLY36

値: 0

No.3

製造者名: ポリテコ

製品名: POLY36

値: 0

名前 Sensor

```
1. public class Sensor {
2.
3.     String productName;
4.     String manufacture;
5.     int value;
6.
7.     public Sensor() {
8.         productName = "POLY35";
9.         manufacture = "ポリテケ";
10.    }
11.
12.    public Sensor(String pn) {
13.        productName = pn;
14.        manufacture = "ポリテケ";
```

```
15.     }
16.
17.     public Sensor(String pn, String m) {
18.         productName = pn;
19.         manufacture = m;
20.     }
21.
22.     public int get() {
23.         return value;
24.     }
25. }
```

名前 TestSensor

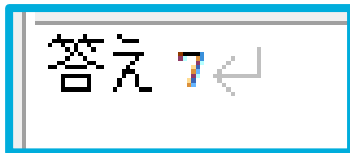
```
1. public class TestSensor {
2.
3.     public static void main(String[] args) {
4.         Sensor[] sensors = new Sensor[3];
5.
6.         Sensor sensor1 = new Sensor();
7.         sensors[0] = sensor1;
8.         Sensor sensor2 = new Sensor("POLY36");
9.         sensors[1] = sensor2;
10.        Sensor sensor3 = new Sensor("POLY36", "ポリテコ");
11.        sensors[2] = sensor3;
12.
13.        info(sensors);
14.    }
15.
16.    public static void info(Sensor[] sensors) {
17.        int i = 1;
18.        for (Sensor s : sensors) {
19.            System.out.println("No." + i);
20.            System.out.println("製造者名: " + s.manufacture);
21.            System.out.println("製品名:   " + s.productName);
22.            System.out.println("値:      " + s.get());
23.            System.out.println();
24.            i++;
25.        }
26.    }
27. }
```

## 例題 14 インスタンスメンバ

- 以下のメンバを持つクラス Calc を作成し、インスタンス化して結果を表示する

クラス名	メンバの種類	データ型 (戻り値の型)	名前	値や振る舞い
Calc	フィールド	int	answer	加算結果がセットされる
	メソッド	void	add(int x,int y)	引数 x、y の合計を求める

実行結果



名前 Calc

```
1. public class Calc {  
2.     int answer;  
3.     void add(int x,int y){  
4.         answer = x + y;  
5.     }  
6. }
```

名前 Reidai14

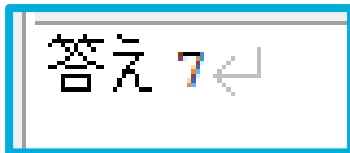
```
1. public class Reidai14 {  
2.     public static void main(String[] args) {  
3.         Calc cal = new Calc();  
4.         cal.add(3,4);  
5.         System.out.println("答え "+ cal.answer);  
6.     }  
7. }
```

## 例題 15 クラスメンバ

- 以下のメンバを持つクラス Calc を作成し、結果を表示する

クラス名	メンバの種類	データ型 (戻り値の型)	名前	値や振る舞い
Calc	フィールド	int	answer	加算結果がセットされる
	メソッド	void	add(int x,int y)	引数 x、y の合計を求める

実行結果



名前 Calc

```
1. public class Calc {  
2.     static int answer;  
3.     static void add(int x,int y){  
4.         answer = x + y;  
5.     }  
6. }
```

名前 Reidai15

```
1. public class Reidai15 {  
2.     public static void main(String[] args) {  
3.         Calc.add(3,4);  
4.         System.out.println("答え "+ Calc.answer);  
5.     }  
6. }
```



## 例題 16 アクセサメソッド

- キーボードから入力した、氏名と点数情報を以下の仕様を持つクラス Student にアクセサメソッドを利用して設定取得を行う

クラス名	メンバの種類	データ型 (戻り値の型)	名前	値
Student	フィールド	String	name	初期値なし
	フィールド	int	tensu	初期値なし
	メソッド	void	setName(String name)	
	振る舞い	引数 name の値をフィールド name に設定		
	メソッド	String	getName()	
	振る舞い	フィールド name の値を取得		
	メソッド	void	setTensu(int tensu)	
	振る舞い	引数 tensu の値をフィールド tensu に設定		
	メソッド	String	getTensu()	
	振る舞い	フィールド tensu の値を取得		

実行結果

※アクセサメソッドの追加手順 (Eclipse)

- ① 新規で Student.java を作成する
- ② クラスのメンバに private でフィールドを宣言する

```

1. public class Student {
2.     private String name;    // 名前格納用
3.     private int tensu;     // 点数格納用
4. }

```

```

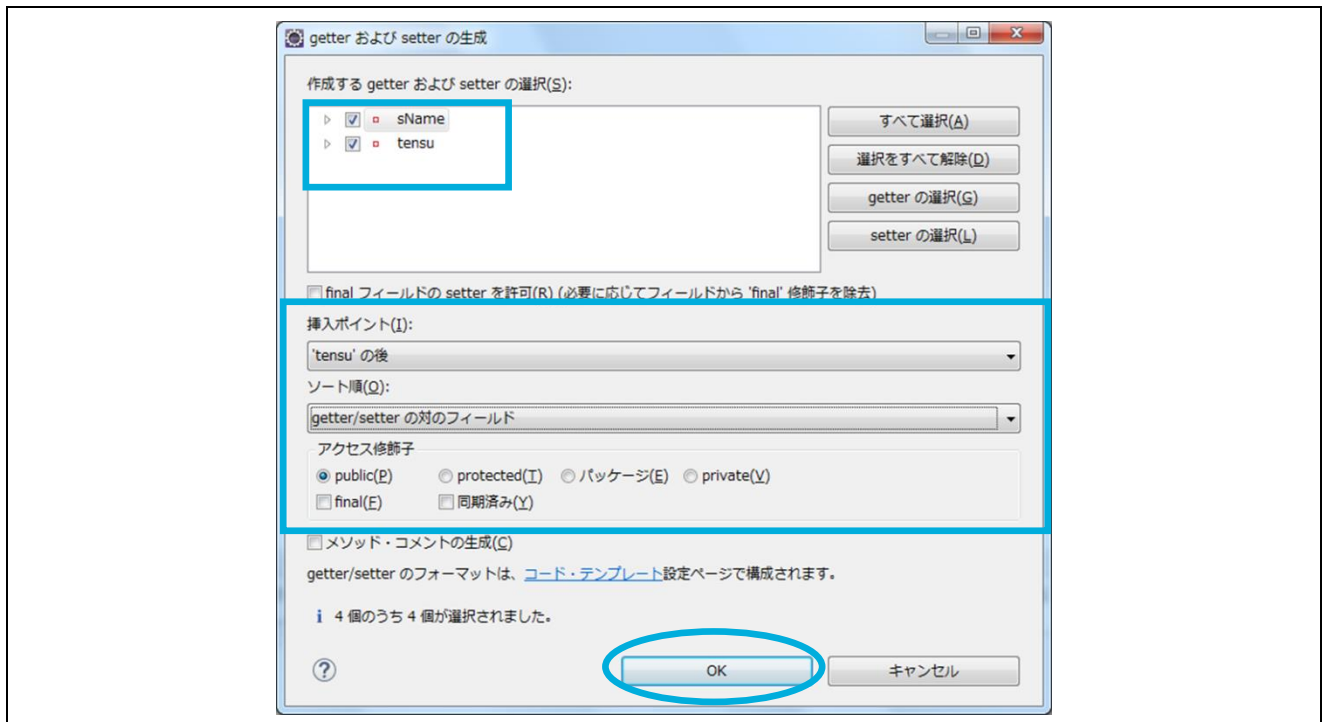
名前を入力:Test<
点数の入力:85<
入力した情報は...<
名前: Test<
点数: 85<

```

- ③ Eclipse のメニュー「ソース」→「getter および setter の生成」を選択。(この時、Eclipse のエディタ画面の対象となるクラス内でカーソルが点滅していること)



- ④ Getter/Setter メソッドを作成したいフィールドに ☒ を入れ、メソッドの挿入位置、並び替え、アクセス制御を設定し、OK を押す



- ⑤ Eclipse では、フィールドだけ入力すれば、機能を使ってアクセサメソッドを自動挿入することができる

```
1. public class Student {  
2.     private String name;  
3.     private int tensu;  
4.     public String getName() {  
5.         return name;  
6.     }  
7.     public void setName(String name) {  
8.         this.name = name;  
9.     }  
10.    public int getTensu() {  
11.        return tensu;  
12.    }  
13.    public void setTensu(int tensu) {  
14.        this.tensu = tensu;  
15.    }  
16. }
```

名前 Student

```
1. public class Student {
2.     private String name;
3.     private int tensu;
4.     public String getName() {
5.         return name;
6.     }
7.     public void setName(String name) {
8.         this.name = name;
9.     }
10.    public int getTensu() {
11.        return tensu;
12.    }
13.    public void setTensu(int tensu) {
14.        this.tensu = tensu;
15.    }
16. }
```

名前 Reidai16

```
1. import java.util.Scanner;
2.
3. public class Reidai16 {
4.
5.     public static void main(String[] args) {
6.         Student str = new Student();
7.         Scanner scan = new Scanner(System.in);
8.         System.out.print("名前の入力:");
9.         str.setName(scan.next());
10.        System.out.print("点数の入力:");
11.        str.setTensu(scan.nextInt());
12.
13.        System.out.println("入力した情報は・・・");
14.        System.out.println("名前: " + str.getName());
15.        System.out.println("点数: " + str.getTensu());
16.    }
17. }
```

## 例題 17 継承

- 以下の仕様を持つクラス temperatureSensor を作成する
- ただし、TemperatureSensor は Sensor クラスを継承する
  - コンストラクタ

引数	振る舞い
String productName	引数 productName の値を親のフィールド productName に代入する
String manufacture	引数 manufacture の値を親のフィールド manufacture に代入する

- フィールド

名前	型	値
temperature	int	温度を格納する

- メソッド

名前	戻り値	引数	振る舞い
get	int	なし	フィールド temperature を返す

名前 Sensor

```
1. public class Sensor {
2.     String productName;
3.     String manufacture;
4.     int value;
5.
6.     public Sensor() {
7.         productName = "POLY35";
8.         manufacture = "ポリテケ";
9.     }
10.
11.    public Sensor(String productName) {
12.        this.productName = productName;
13.        manufacture = "ポリテケ";
14.    }
15.
16.    public Sensor(String productName, String manufacture) {
17.        this.productName = productName;
18.        this.manufacture = manufacture;
19.    }
20.
21.    public int get() {
22.        return value;
23.    }
24. }
```

名前 TemperatureSensor

```
1. public class TemperatureSensor extends Sensor {
2.     int temperature;
3.
4.     public TemperatureSensor(String productName, String manufacture) {
5.         this.productName = productName;
6.         this.manufacture = manufacture;
7.     }
8.
9.     public int get() {
10.        return temperature;
11.    }
12. }
```

名前 TestTemperatureSensor

```
1. public class TestTemperatureSensor {
2.     public static void main(String[] args) {
3.         TemperatureSensor ts =
4.             new TemperatureSensor("POLY35", "ポリテケ");
5.
6.         System.out.println("製造者名: " + ts.manufacture);
7.         System.out.println("製品名:   " + ts.productName);
8.         System.out.println("温度:     " + ts.get());
9.     }
10. }
```

## 例題 18 アップキャストとダウンキャスト

- 例題 17 の TemperatureSensor クラスの仕様にメソッドを追加し、アップキャストとダウンキャストを確認する

名前	戻り値	引数	振る舞い
getFahrenheit	int	なし	華氏を返す
getCelsius	Int	なし	摂氏を返す

名前 Sensor (例題 17 から変更なし)

```
1. public class Sensor {
2.     String productName;
3.     String manufacture;
4.     int value;
5.
6.     public Sensor() {
7.         productName = "POLY35";
8.         manufacture = "ポリテケ";
9.     }
10.
11.    public Sensor(String productName) {
12.        this.productName = productName;
13.        manufacture = "ポリテケ";
14.    }
15.
16.    public Sensor(String productName, String manufacture) {
17.        this.productName = productName;
18.        this.manufacture = manufacture;
19.    }
20.
21.    public int get() {
22.        return value;
23.    }
24. }
```

名前 TemperatureSensor

```
1. public class TemperatureSensor extends Sensor {
2.     int temperature;
3.
4.     public TemperatureSensor(String productName, String manufacture) {
5.         this.productName = productName;
6.         this.manufacture = manufacture;
7.     }
8.
9.     public int get() {
10.        return temperature;
11.    }
12.
13.    public int getFahrenheit() {
14.        double t = temperature * 1.8 + 32.0;
15.        return (int)t;
16.    }
17.
18.    public int getCelsius() {
```

```
19.         return temperature;
20.     }
21. }
```

名前 TestTemperatureSensor

```
1. public class TestTemperatureSensor {
2.
3.     public static void main(String[] args) {
4.         TemperatureSensor ts =
5.             new TemperatureSensor("POLY35", "ポリテケ");
6.
7.         Sensor sensor = new TemperatureSensor("POLY35", "ポリテケ");
8.         System.out.println("製造者名: " + sensor.manufacture);
9.         System.out.println("製品名:   " + sensor.productName);
10.        System.out.println("温度:     " + sensor.get());
11.        //System.out.println("温度[F]: " + sensor.getFahrenheit());
12.
13.        TemperatureSensor tSensor = (TemperatureSensor)sensor;
14.        System.out.println("製造者名: " + tSensor.manufacture);
15.        System.out.println("製品名:   " + tSensor.productName);
16.        System.out.println("温度[C]:  " + tSensor.get());
17.        System.out.println("温度[F]:  " + tSensor.getFahrenheit());
18.    }
19. }
```

## 例題 19 オーバーロード

- 例題 18 の TemperatureSensor クラスの仕様にメソッドを追加して、オーバーロードの確認をする

戻り値	名前	説明
int	get(int unit)	引数の種類に依り、摂氏と華氏を返す
unit が TemperatureSensor.Celsius のときは摂氏を返す		
unit が TemperatureSensor.Fahrenheit のときは華氏を返す		

実行結果

製造者名: ポリテケ

製品名: POLY35

温度[C]: 0

温度[F]: 32

名前 Sensor

```
1. public class Sensor {
2.     String productName;
3.     String manufacture;
4.     int value;
5.
6.     public Sensor() {
7.         this("POLY35", "ポリテケ");
8.     }
9.
10.    public Sensor(String productName) {
11.        this(productName, "ポリテケ");
12.    }
13.
14.    public Sensor(String productName, String manufacture) {
15.        this.productName = productName;
16.        this.manufacture = manufacture;
17.    }
18.
19.    public int get() {
20.        return value;
21.    }
22. }
```

名前 TemperatureSensor

```
1. public class TemperatureSensor extends Sensor {
2.     static final int Fahrenheit = 1;
3.     static final int Celsius = 2;
4.     int temperature;
```



```

5.
6.     public TemperatureSensor() {
7.         super();
8.     }
9.
10.    public TemperatureSensor(String productName, String manufacture) {
11.        super(productName, manufacture);
12.    }
13.
14.    public int get() {
15.        return temperature;
16.    }
17.
18.    public int get(int unit) {
19.        if (unit == Fahrenheit) {
20.            return getFahrenheit();
21.        } else if (unit == Celsius) {
22.            return getCelsius();
23.        } else {
24.            return -9999;
25.        }
26.    }
27.
28.    public int getFahrenheit() {
29.        double t = temperature * 1.8 + 32.0;
30.        return (int)t;
31.    }
32.
33.    public int getCelsius() {
34.        return temperature;
35.    }
36. }

```

名前 TestTemperatureSensor

```

1. public class TestTemperatureSensor {
2.     public static void main(String[] args) {
3.         TemperatureSensor ts =
4.             new TemperatureSensor("POLY35", "ポリテケ");
5.
6.         System.out.println("製造者名: " + ts.manufacture);
7.         System.out.println("製品名: " + ts.productName);

```

```
8.      System.out.println("温度[C]:  " +
9.          ts.get(TemperatureSensor.Celsius));
10.     System.out.println("温度[F]:  " +
11.         ts.get(TemperatureSensor.Fahrenheit));
12.     }
13. }
```

## 例題 20 オーバーライド

- 例題 18 の仕様にメソッドを追加して、オーバーライドの確認をする

クラス名	戻り値	名前
Sensor	String	toString()
	クラス名、製造者と製品名を含んだ文字列を生成	
TemperatureSensor	String	toString()
	クラス名、製造者と製品名を含んだ文字列を生成	

実行結果

製造者名: ポリテケ

製品名: POLY35

温度[C]: 0

温度[F]: 32

TemperatureSensor Class 製品名: POLY35, 製造者: ポリテケ

名前 Sensor

```
1. public class Sensor {
2.
3.     String productName;
4.     String manufacture;
5.     int value;
6.
7.     public Sensor() {
8.         this("unknown", "unknown");
9.     }
10.
11.    public Sensor(String productName) {
12.        this(productName, "unknown");
13.    }
14.
15.    public Sensor(String productName, String manufacture) {
16.        this.productName = productName;
17.        this.manufacture = manufacture;
18.    }
19.
20.    public int get() {
21.        return value;
22.    }
23.
24.    public String toString() {
25.        return "Sensor Class 製品名: " + productName +
26.            ", 製造者: " + manufacture;
27.    }
28. }
```

名前 TemperatureSensor

```
1. public class TemperatureSensor extends Sensor {
2.
3.     static final int Fahrenheit = 1;
4.     static final int Celsius = 2;
5.
6.     int temperature;
```

```

7.
8.     public TemperatureSensor() {
9.         super();
10.    }
11.
12.    public TemperatureSensor(String productName, String manufacture) {
13.        super(productName, manufacture);
14.    }
15.
16.    public int get() {
17.        return temperature;
18.    }
19.
20.    public int get(int unit) {
21.        if (unit == Fahrenheit) {
22.            return getFahrenheit();
23.        } else if (unit == Celsius) {
24.            return getCelsius();
25.        } else {
26.            return -9999;
27.        }
28.    }
29.
30.    public int getFahrenheit() {
31.        double t = temperature * 1.8 + 32.0;
32.        return (int)t;
33.    }
34.
35.    public int getCelsius() {
36.        return temperature;
37.    }
38.
39.    public String toString() {
40.        return "TemperatureSensor Class 製品名: " + productName +
41.            ", 製造者: " + manufacture;
42.    }
43. }

```

名前 TestTemperatureSensor

```

1. public class TestTemperatureSensor {
2.

```

```
3.     public static void main(String[] args) {
4.         TemperatureSensor ts =
5.             new TemperatureSensor("POLY35", "ポリテケ");
6.
7.         System.out.println("製造者名: " + ts.manufacture);
8.         System.out.println("製品名:   " + ts.productName);
9.         System.out.println("温度[C]:  " +
10.            ts.get(TemperatureSensor.Celsius));
11.         System.out.println("温度[F]:  " +
12.            ts.get(TemperatureSensor.Fahrenheit));
13.         System.out.println(ts.toString());
14.     }
15. }
```

## 例題 21 抽象クラス

- Sensor クラスを抽象クラスとし、以下の抽象メソッドを追加する。

戻り値	名前
int	get()

名前 Sensor

```
1. public abstract class Sensor {
2.     String productName;
3.     String manufacture;
4.     int value;
5.
6.     public Sensor() {
7.         this("unknown", "unknown");
8.     }
9.
10.    public Sensor(String productName) {
11.        this(productName, "unknown");
12.    }
13.
14.    public Sensor(String productName, String manufacture) {
15.        this.productName = productName;
16.        this.manufacture = manufacture;
17.    }
18.
19.    public abstract int get();
20. }
```

## 例題 22 インターフェース

- 以下の仕様を持つクラスやインターフェースを定義し、動作確認をする

クラス名 又は インターフェース名	メンバの種類	データ型 (戻り値の型)	名前	値
Shape (インターフェース)	定数	double	PI	3.141592
	抽象メソッド	double	getArea()	
Rect (Shape を実装)	フィールド	double	width	
	フィールド	double	height	
	コンストラクタ	なし	Rect(double width,double height)	
	振る舞い	引数 width,height の値をフィールドに代入		
	メソッド (オーバーライド)	double	getArea()	
	振る舞い	width と height の幅,高さを持つ四角形の面積を求める		
Circle (Shape を実装)	フィールド	double	radius	
	コンストラクタ	なし	Circle(double radius)	
	振る舞い	引数 radius の値をフィールドに代入		
	メソッド (オーバーライド)	double	getArea()	
	振る舞い	半径が radius の円の面積を求める		

実行結果

```
10×5の四角形の面積: 50.0<
半径3の円の面積: 28.274328<
```

名前 Shape

```
1. public interface Shape {
2.     static final double PI = 3.141592;
3.     public double getArea();
4. }
```

名前 Rect

```
1. public class Rect implements Shape {
2.     double width;
3.     double height;
4.     public Rect(double width, double height) {
5.         this.width = width;
6.         this.height = height;
7.     }
8.     @Override
9.     public double getArea() {
10.         return width*height;
11.     }
12. }
```

```
11.     }  
12. }
```

名前 Circle

```
1. public class Circle implements Shape {  
2.     double radius;  
3.     public Circle(double radius) {  
4.         this.radius = radius;  
5.     }  
6.     @Override  
7.     public double getArea() {  
8.         return radius*radius*PI;  
9.     }  
10. }
```

名前 Reidai22

```
1. public class Reidai22 {  
2.     public static void main(String[] args) {  
3.         Rect rect1 = new Rect(10.0,5.0);  
4.         System.out.println("10×5 の四角形の面積：" + rect1.getArea());  
5.         Circle cir1 = new Circle(3.0);  
6.         System.out.println("半径 3 の円の面積：" + cir1.getArea());  
7.     }  
8. }
```



## 例題 23 例外処理

- キーボードから2つの数値を入力して、除算した結果を表示する。
- ただし、以下の条件のとき例外を発生させて対応する

条件	例外名	対応処理
数値でない値をキーボードから入力した	<code>InputMismatchException</code>	数値を入力してくださいとメッセージを表示し終了
0 で除算してしまった	<code>ArithmeticException</code>	0 で割っていますとメッセージを表示し終了

実行結果

名前 Reidai23

```

1. import java.util.InputMismatchException;
2. import java.util.Scanner;
3.
4. public class Reidai23 {
5.     public static void main(String[] args) {
6.         int[] input = new int[2];
7.         int cnt=0;
8.         try{
9.             Scanner scan = new Scanner(System.in);
10.            do{
11.                System.out.print("数値を入力 -> ");
12.                input[cnt] = scan.nextInt();
13.                cnt++;
14.            }while(cnt < 2);
15.        }catch(InputMismatchException e){
16.            System.err.println("数値を入力してください");
17.            e.printStackTrace();
18.            System.exit(0);
19.        }
20.
21.        try{
22.            double answer = input[0] / input[1];
23.            System.out.println("結果は・・・"+answer);
24.        }catch(ArithmeticException e){
25.            System.err.println("0 で割ってます");
26.            e.printStackTrace();
27.            System.exit(0);
28.        }
29.    }
30. }

```

```

数値を入力 -> g
数値を入力してください
java.util.InputMismatchException
数値を入力 -> 10
数値を入力 -> 0
0で割ってます
java.lang.ArithmeticException: / by zero
    at reidai25.Reidai25.main(Reidai25.java:25)

```

```

数値を入力 -> 10
数値を入力 -> 2
結果は・・・5.0

```

## 例題 24 匿名クラス

- 以下のような仕様を持つクラスを作成し、動作確認をする

クラス名	メンバの種類	データ型 (戻り値の型)	名前	値
Shape	フィールド	double	width	
	フィールド	double	height	
	メソッド	void	getArea()	
	振る舞い	width と height の幅,高さを持つ四角形の面積を求めて表示		
匿名クラス内 (Shape クラスのサブクラス)	フィールド	double	radius	3.0
	メソッド	void	getArea()	
	振る舞い	半径 radius の円の面積を求めて表示		

実行結果

名前 Reidai24

```

1. public class Reidai24 {
2.
3.     public static void main(String[] args) {
4.         Shape shape = new Shape(){
5.             double radius = 3.0;
6.             final double PI = 3.141592;
7.             void getArea(){
8.                 System.out.println("半径 3.0 の円の面積は : "+ radius*radius*PI);
9.             }
10.        };
11.
12.        shape.getArea();
13.
14.        Shape orgShape = new Shape();
15.        orgShape.getArea();
16.    }
17. }
18. class Shape{
19.     double width=10.0;
20.     double height = 5.0;
21.     void getArea(){
22.         System.out.println("10×5 の四角形の面積は : "+ width*height);
23.     }
24. }
```

```

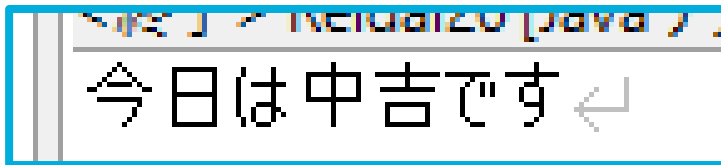
<終了> Reidai24 [Java アプリケーション] C:\pleiade
半径3.0の円の面積は: 28.274328<
10×5の四角形の面積は: 50.0<
```

## 例題 25 列挙型

- 以下の仕様を持つ列挙型を生成し、おみくじアプリケーションを作成する

項目	設定値
列挙型名	Omikuji
列挙子リスト	DAIKICHI,CHUKICHI,KICHI,KYO,DAIKYO

実行結果



名前 Reidai25

```
1. import java.util.Random;
2.
3. public class Reidai25 {
4.
5.     enum Omikuji{
6.         DAIKICHI,CHUKICHI,KICHI,KYO,DAIKYO;
7.     };
8.
9.     public static void main(String[] args) {
10.         Random rdm = new Random();
11.         int mikuji = rdm.nextInt(5);
12.         if(Omikuji.DAIKICHI.ordinal() == mikuji){
13.             System.out.println("今日は大吉です");
14.         }else if(Omikuji.CHUKICHI.ordinal() == mikuji){
15.             System.out.println("今日は中吉です");
16.         }else if(Omikuji.KICHI.ordinal() == mikuji){
17.             System.out.println("今日は吉です");
18.         }else if(Omikuji.KYO.ordinal() == mikuji){
19.             System.out.println("今日は凶です");
20.         }else if(Omikuji.DAIKYO.ordinal() == mikuji){
21.             System.out.println("今日は大凶です");
22.         }
23.     }
24. }
```

## 例題 26 コレクションクラス

- クラス、コレクションクラスを作成して、データを格納して動作を確認する

項目	設定値
クラス名	City
フィールド	String nameofcity
フィールド	double longitude
フィールド	double latitude

City 型インスタンス city に格納するデータ

項目	設定値
city[0].nameofcity	東京
city [0].longitude	139.766084
city [0].latitude	35.681382
city[1].nameofcity	ロンドン
city[1].longitude	-0.127758
City[1].latitude	51.507351

City 型 ArrayList のインスタンス array に追加するデータ

city[0]	city[1]
---------	---------

キーString, 値 City 型 HashMap のインスタンス hash に追加するデータ

(キー, 値)=(“Tokyo”, city[0])    (キー, 値)=(“London”, city[1])

表示する項目

ArrayList インスタンスに格納した City クラスの nameofcity フィールド情報

HashMap インスタンスに格納したキー「Tokyo」の値 nameofcity フィールド情報

実行結果

```
<終了> TestException [Java  
ArrayList [0]=東京  
ArrayList [1]=ロンドン  
HashMap Tokyo=東京
```

名前 Reidai26

```
1. import java.util.ArrayList;  
2. import java.util.HashMap;  
3. import java.util.List;  
4. import java.util.Map;  
5.  
6. public class Reidai26 {  
7.
```

```

8.     public static void main(String[] args) {
9.         List<City> array = new ArrayList<City>();
10.        Map<String, City> hash = new HashMap<String, City>();
11.
12.        // データの追加
13.        City[] city = new City[2];
14.        for(int i = 0; i < city.length ; i++){
15.            city[i] = new City();
16.        }
17.        city[0].nameofcity = "東京";
18.        city[0].longitude = 139.766084;
19.        city[0].latitude = 35.681382;
20.        city[1].nameofcity = "ロンドン";
21.        city[1].longitude = -0.127758;
22.        city[1].latitude = 51.507351;
23.
24.        array.add(city[0]);
25.        array.add(city[1]);
26.
27.        hash.put("Tokyo", city[0]);
28.        hash.put("London", city[1]);
29.
30.        // 表示
31.        for(int i = 0 ; i < array.size() ; i++){
32.            System.out.println("ArrayList[" + i + "]="+ array.get(i).nameofcity);
33.        }
34.        System.out.println("HashMap Tokyo=" + hash.get("Tokyo").nameofcity);
35.    }
36. }
37.
38. class City{
39.     String nameofcity;    // 都市名
40.     double longitude;      // 経度
41.     double latitude;      // 緯度
42. }

```