

## 目次

### Part1 Java プログラミングの基本 p. 2

- 1.はじめの一步
- 2.データと演算
- 3.変数
- 4.乱数
- 5.練習1
- 6.if 文と条件式
- 7.while 文
- 8.for 文
- 9.練習2
- 10.メソッド
- 11.練習2の解答例

### Part2 グラフィックス p. 14

- 1.骨組みを作る
- 2.図形の描画
- 3.色
- 4.変数を使ったアニメーション
- 5.クラス①
- 6.クラス②
- 7.リストクラス
- 8.クラスの継承
- 9.練習
- 10.キー操作

### Part3 シューティングゲームを作ろう p. 27

- 1.シューティングゲームの説明
- 2.骨組みを作る
- 3.画面上の物体:Item クラス
- 4.弾:Bullet クラス
- 5.機体:Kitai クラス
- 6.敵:Enemy クラス
- 7.キー操作:Keydata クラス
- 8.自機:Player クラス
- 9.ゲーム本体:Shooting クラス
- 10.改良しよう

### Part4 シューティングゲームを改良しよう p. 41

- 1.勝敗を判定し、表示する
- 2.タイトル画面、ゲームオーバー画面を作る
- 3.敵の攻撃パターンを増やす
- 4.画像を表示する
- 5.敵の情報をファイルから読み込む
- 6.マップの情報をファイルから読み込む

### 付録: 環境設定 p. 52

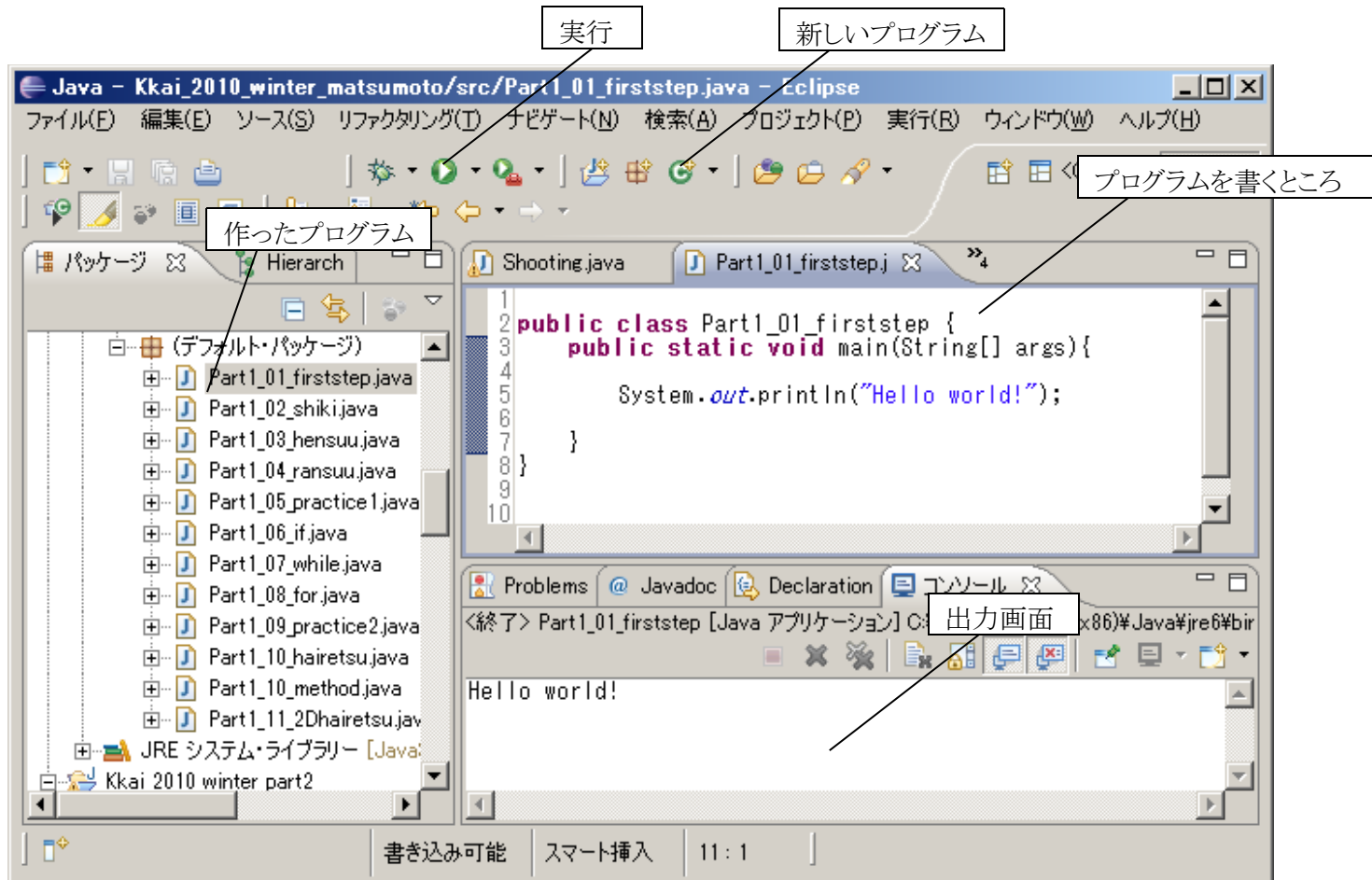
## Part1 Java プログラミングの基本

まずはプログラムを書くことに慣れましょう。

1. はじめの一步
2. データと演算
3. 変数
4. 乱数
5. 練習 1
6. if 文と条件式
7. while 文
8. for 文
9. 練習 2
10. メソッド
11. 練習 2 の解答例

## 1. はじめの一步

早速プログラムを作ってみましょう。ファイルを作成して、プログラムを書いて、実行するという手順は、これから何十回と繰り返すことになります。ここでしっかりと覚えておきましょう。



### ☆プログラム作成の手順

- ① をクリック→名前を「Part1\_01\_firststep」として OK を押す
- ② 上のとおりにプログラムを入力する
- ③ バグ(間違い)があれば赤線が引かれるので直す(デバッグといいます)
- ④ を押して実行！(変な画面が出てきたらとりあえず OK を押す)
- ⑤ うまいければ出力画面に「Hello world!」と出てきます

### ☆ポイント

- ① 大文字小文字、記号に注意。
- ② 行の先頭の空白は「Tab キー」です。スペースキーで入力するとずれるので気をつけて下さい。
- ③ `System.out.println( );` は、カッコの中身を 1 行に出力しろ、という命令です。

### ☆便利なショートカットキー

コピー: Ctrl キーを押しながら「C」キーを押す  
 切り取り: Ctrl キーを押しながら「X」キーを押す  
 貼りつけ: Ctrl キーを押しながら「V」キーを押す  
 元に戻す: Ctrl キーを押しながら「Z」キーを押す(みすったときに)

## 2. データと演算

人間は言葉や数を使って、考えたり、会話したりします。コンピュータも「データ」を使って計算したり、何かを表示したりします。

プログラムが扱うデータは、例えば次のようなタイプがあります。

①整数(int)

例: 123、-1

②小数(double)

例: 0.5、3.141592

③真偽値(boolean)

「true」(真)と「false」(偽)の二通りだけです。

④文字(char)

例: a,b,c,~,!,@,#,\$ 等の文字、記号

これらのデータを使った計算を行うことができます。ここでは整数と小数の計算をさせてみましょう。

整数、小数は、おなじみの足し算、引き算、掛け算、割り算ができます。割った余りを求める、という計算もできます。記号は以下のものを使います。

足し算: +

引き算: -

掛け算: \*

割り算: /

割った余り: %

ここで、新しいプログラム「Part1\_02\_data」をさっきと同じ手順で作りましょう。先ほど書いたプログラムをコピーして、System.out.print();の部分を書き換えるのが楽です。

System.out.println(1+1); のように、カッコの中に次の数式を書いて実行してみましょう。

1+3\*4/2      足し算より掛け算、割り算の方が先に行われます。電卓とは違うので注意。

(1+3)\*4/2      ()でくくると、そこが先に計算されます。算数で習ったのと同じですね。

5/2      整数と整数を割り算すると整数になります。結果は切り捨てです。

5.0/2.0      小数と小数を割り算すれば結果は小数になります。

30%4      割った余りを求めるには[%]をつかいます。

他にもいろんな数式を入れて試してみましょう。もし赤線が引かれたら、何かが間違っています。

また、コンピュータは大きすぎる数字を扱えません。だから 123456789\*987654321 のような計算をさせると間違えます。

### 3. 変数

データを記録しておく箱のようなものを作ることができます。これを変数といいます。変数は次の手順で使います。

#### ①宣言する

この命令を書くことで、変数が作られます。

```
int hensuu;
```

```
double d;
```

のように、タイプの後に、変数の名前を言います。変数の名前は英数字数文字の好きな名前がつけられます(ただし、いくつかの単語は紛らわしいため使えません。こういった単語を使うと赤線が引かれます)

#### ②代入する

変数にデータをいれることを代入といいます。代入した後は、次に代入されるまで、その変数は代入したデータそのもののようふるまいます。

代入には「=」記号を使います。右のデータを左の変数に代入するという意味です。

```
hensuu = 100;
```

```
d = 3.14;
```

double d2 = d; このように、宣言と代入を同時にすることができます。また、変数を変数に代入できる。

#### ③計算する

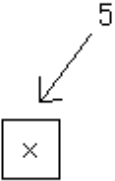
先程の「データ」と同じように、変数同士を足し算したり、変数を出したりできます。実際にプログラムを書いてみましょう。

新しいプログラム「Part1\_03\_hensuu」を作りましょう。

```

1 public class Part1_03_hensuu {
2     public static void main(String[] args) {
3
4         int x;                //整数型の変数xを定義する
5         x = 5;                //xに5を入れる
6         x = x / 2;            //xにx/2を入れる
7         x = x * x;            //xにx*xを入れる
8         System.out.println(x); //何が出力されますか?
9
10        double y;             //小数型の変数yを定義する
11        y = 1.0;              //yに1.0を入れる
12        y = y / 2.0;          //yにy/2.0を入れる
13        y = y * y;            //yにy*yを入れる
14        System.out.println(y); //何が出力されますか?
15
16        System.out.println(x*y); //何が出力されますか?(整数と小数の掛け算は結果は小数)
17    }
18 }
19

```



#### ☆注意

- 一度宣言した変数と同じ名前の変数をもう一度宣言することはできない。
- 次の省略記法が便利
  - $x = x + 1;$  →  $x++;$  (1を足す)
  - $y = y - 1;$  →  $y--;$  (1を引く)
  - $z = z + a;$  →  $z += a;$  (aを足す。-,\*,/,%でも同様にできる)

#### 4. 乱数

ゲームには偶然の要素が付き物。プログラムで「偶然」を作る方法、それが乱数です。  
 使い方は簡単で、`Math.random()`と書くだけで、0～1.0 までのランダムな小数が得られます。サイコロを振るようなものです。

変数の練習も兼ねて、新しいプログラム「Part1\_04\_ransuu」を作しましょう。

```

1
2 public class Part1_04_ransuu {
3     public static void main(String[] args){
4
5         double a = Math.random();
6         double b = Math.random();           //aとbをランダムな数にする
7
8         System.out.println(a + " * " + b);   //この+はちょっと特別で、
9         System.out.println(a*b);             //文字と文字をつなげるという意味になります。
10
11         a = Math.random();
12         b = Math.random();                   //もう一度aとbにランダムな数をいれます。
13
14         System.out.println(a + " * " + b);
15         System.out.println(a*b);             //結果は変わります
16     }
17 }
18
19

```

`Math.random()`がたくさんでできますが、コピー & 貼りつけを使って手早く入力しましょう。  
`Math.random()`が代入するたびに違う値になることがわかります。また、実行するたびに値は変わります。

#### 5. 練習 1

次のようなプログラムを作りましょう。(ファイル名は Part1\_05\_practice1 などにしましょう)。

- ① 7373\*1507 を計算せよ
- ② 1.2345 の5乗(5回掛けたもの)を、変数を使って計算せよ
- ③ 0 から 9 までのランダムな**整数**を表示せよ。ただし、小数を整数に切り捨てるには、  
(int)(切り捨てたい小数)  
と書く。  
ヒント: Math.random() を 10 倍すると・・・?
- ④ 以下のプログラムの空白を埋めて、一行目と二行目で、左右に出力されるものが入れ替わっているようにせよ。  
(たとえば、1 行目が 0.12345 0.98765 なら、2 行目は 0.98765 0.12345)

```
double x = Math.random();  
double y = Math.random();  
System.out.println(x + " " + y);
```

```
System.out.println(x + " " + y);
```

ヒント: 単純に、

x=y;

y=x;

と書くとうまくいかない。なぜか?

## 6. if 文と条件式

「もし…だったら～～せよ。」という命令をプログラムに書くことができます。

```
if(...){
    ～～
}
```

…の部分には、データの章ででてきた「真偽値」true(正しい)または false(間違い)が入ります。true ならば、中括弧{}の中身である「～～」が実行されます。しかし「true」そのものを書いて (if(true){ })、常に{}が実行されるだけで意味が無いので、合ってるか間違ってるか判断できる文、「条件式」を書くことになります。

条件式には次のようなものがあります。

```
a == b   a と b が等しい
a != b   a と b が等しくない
a < b    a が b より小さい
a <= b   a が b 以下
a > b    a が b より大きい
a >= b   a が b 以上
```

```
(a==b) && (c==d)   a が b と等しく、かつ、c が d と等しい
(a==b) || (c==d)   a が b と等しい、または、c が d と等しい
```

これらの条件式は、式が正しい時に true、正しくないときに false になります。

3<5 は true、10!=10 は false です。

また、「そうでなければ」の意味の else という単語を使って、次のような書き方ができます。

```
if(A){
    A が true の時実行される
}else if(B){
    A が false で、B が true の時実行される
}else{
    A も B も false の時実行される
}
```

実際にプログラムを書いてみましょう。「Part1\_06\_if」というファイルを作ってください。

```
1
2 public class Part1_06_if {
3     public static void main(String[] args){
4
5         double r = Math.random();
6         System.out.println(r);
7
8         if (0.3 <= r && r < 0.7){           //rが0.3以上0.7未満ならば
9             System.out.println("OK");
10        }else if (r < 0.3){                 //そうでなくて、0.3未満ならば
11            System.out.println("Small");
12        }else{                             //そうでもなければ（つまり0.7以上）
13            System.out.println("Big");
14        }
15    }
16 }
17 }
```

中括弧{}を開いたら、Tab キー一つ分右にずらすのを忘れないでください。こうすることで、プログラムがだいぶ読みやすくなります。

## 7. while 文



同じことを何回も繰り返すのは、人間は嫌になりますが、コンピュータは大得意です。  
「A が true であるかぎり～～～せよ。」というのは以下のように書きます。

```
while(A){
    ～～～
}
```


一回～～～を実行するたびに、A が true かどうか調べられ、true なら再び～～～を実行します。  
もし A が false にならないと、～～～を無限に繰り返し、プログラムは終了しません。これは「無限ループ」といって、よくあるミスのひとつです。

「Part1\_07\_while」というファイルを作って、以下のように書いてください。

```
1 public class Part1_07_while {
2     public static void main(String[] args){
3
4         int i = 0;
5         while(i < 10){                //iが10未満である限り{}の中身を繰り返す
6             System.out.println(i);    //i = i + 1 の省略記法
7             i++;
8         }
9     }
10 }
11
12
13
```

繰り返しが起こるたびに変数 i に 1 が足されるので、10 回繰り返した時点で i が 10 より大きくなり、ループが中断されます。

while(i<10)を while(i>=0)に書き換えて、無限ループさせてみましょう。変数 i は最初 0 で、大きくなる一方なので、{}の中身が無限に繰り返されます。

無限ループしているプログラムの止め方は、画面の下の方にある  ボタンを押してください。PC の調子が悪くなることもあるのでほどほどにしましょう。

ちなみに、ループを抜けろという意味の命令 break; があります。次のように書くと、上のプログラムと同じように動きます。

```
int i = 0;
while(true){                //このままだと無限に繰り返すが、
    if(i >= 10) break;       //こう書けばiが10以上になったらループを抜ける
    System.out.println(i);
    i++;
}
```

さらにちなみに、if文などは、{}の中身が一行だけならば、{}を省略できます。

この例の if(i>=10) break;はこの規則にしたがって書かれています。プログラムを簡潔にかけるので、便利です。

## 8. for 文

繰り返しの処理はとてもよく使うので、while 文よりも簡潔に書く方法があります。

```
for(int i=0; i<10; i++){
    ~~~
}
```

と書くと、

```
int i=0;
while(i<10){
    ~~~
    i++;
}
```

と書くのと同じ意味になります。2 行分の節約ができます。どの文が、for の中身のどの位置に配置されているかよく見てみてください。

```
for(A; B; C){
    ~~~
}
```

は、「一番最初に A を実行し、B が true であるかぎり {} の中身を実行し、一回繰り返すたびに C を実行しろ。」という意味になります。

繰り返しの処理に慣れるという意味合いも込めて、次のプログラム「Part1\_08\_for」を作しましょう。

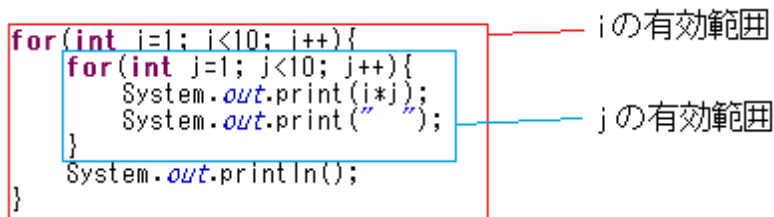
```
1 public class Part1_08_for {
2     public static void main(String[] args){
3
4         for(int i=1; i<10; i++){
5             for(int j=1; j<10; j++){
6                 System.out.print(i*j);           //System.out.print()は改行をしない出力
7                 System.out.print(" ");           //この空白はスペースではなくてTabです
8             }                                     //ここで改行する
9             System.out.println();
10        }
11    }
12 }
13 }
14 }
15 }
```

九九の表が出力されましたか？このように、ループの中にループを書くこともできます。

今までやってきた、if、while、for は、いくつも入れ子にすることが出来ます。

#### ☆注意

変数には有効範囲があります。中括弧の中、あるいはその手前の for の中身で宣言した変数は、その中括弧の中だけで有効です。したがって、中括弧を閉じた後ではその変数は消滅してしまいます。その反面、同じ名前の変数をもう一回宣言することもできます。



```
for(int i=1; i<10; i++){
    for(int j=1; j<10; j++){
        System.out.print(i*j);
        System.out.print(" ");
    }
    System.out.println();
}
```

iの有効範囲

jの有効範囲

## 9. 練習 2

次のようなプログラムを作りましょう。(ファイル名は Part1\_09\_practice2 などにしましょう)。  
結構難しいので、できるところまで結構です。

① 1 から 10000 までの和を計算せよ

ヒント: 和を記録する変数を作る

② 11111111 の約数をすべて出力せよ

ヒント: 「11111111 が  $a$  で割り切れる」は、「 $11111111 \% a == 0$ 」と書ける。

③ 前二つの和が次の数になってるような数列(1 1 2 3 5 8 13...)を 20 番目まで出力せよ

(これをフィボナッチ数列という)

④ 1 から 100 までの、3 の倍数ではなくて、かつ、3 が含まれない数を出力せよ。

1 2 4 5 7 8 10 11 14...

ヒント: 1 の位は単に 10 で割った余りをとればよい。10 の位は、10 で割ってから 10 で割った余りをとれば得られる。(314 を 10 で割ると 4 は切り捨てられるので 31 となるから、10 で割った余りを取ると十の位の数だった 1 になる)

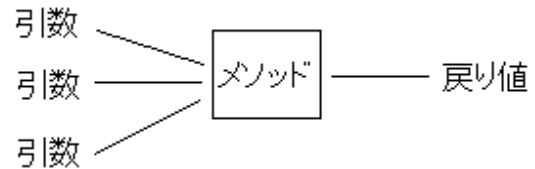
⑤ 1 から 1000 までの素数(1 とそれ自身以外に約数を持たない数)を全て求めよ

プログラムを書いていると、同じような文を何回も書かないといけないようなことがあります。こういうときは、「メソッド」にしてまとめてしまうことができます。

メソッドには、いくつかの「引数」と一つの「戻り値」があります。メソッドは引数のデータに何らかの計算をして、その結果として戻り値を返します。例えば、下の例の max メソッドは、引数として二つの整数型の変数を取り、そのうちの大きい方の値を戻り値として返します。

#### ①メソッドの定義

```
修飾子 戻り値の型 メソッド名(引数1, 引数2, ...){
    処理
    return 戻り値;
}
```



修飾子については、もう少し後で、class の話が出てきたところで説明します。

今まで何気なく書いていた `public static void main(String[] args){ }` も、実は main という特別なメソッドだったのです。ですからメソッドを書く場所は、main メソッドの外側です。

引数と戻り値は、なくても構いません。戻り値がないときは、void と書きます。

#### ②メソッドの使い方

引数が 2 つ、戻り値が int の method という名前のメソッドがあったとしたら、

```
int a = method(x,y);
```

のように、変数 x と y を引数として与えて、その結果の戻り値を a に代入する、というふうに使えます。

「Part1\_10\_method」を作りましょう。この例を使ってメソッドの使い方を体で覚えてしまいましょう。それぞれのメソッド(max、sayhello、hypot、intrandom)が何を意味しているか考えながらプログラミングしてください。

```

1
2 public class Part1_10_method {
3
4     static int max(int a, int b){
5         if(a>b) return a;
6         else return b;
7     }
8
9     static void sayhello(int n){
10        for(int i=0; i<n; i++){
11            System.out.println("Hello!");
12        }
13    }
14
15    static double hypot(double a, double b){
16        return Math.sqrt(a*a + b*b); //Math.sqrt()は、()の中身のルートを返すメソッドです
17    }
18
19    static int intrandom(){
20        return (int)(Math.random()*100);
21    }
22
23    public static void main(String[] args) {
24        int a = 3;
25        int b = 4;
26        int c = max(a,b);
27        sayhello(c);
28        System.out.println(hypot(a,b));
29        System.out.println(intrandom());
30    }
31 }
32 }
33
```

## 1 1. 練習 2 の解答例

```

//-----①-----//
int a = 0;
for(int i=0; i<=10000; i++){
    a = a + i;
}
System.out.println(a);

//-----②-----//
for(int i=1; i<=11111111; i++){
    if(11111111 % i == 0){
        System.out.println(i);
    }
}

//-----③-----//
int a1 = 1;
int a2 = 1;
for(int i=0; i<20; i++){
    System.out.println(a1);
    int t = a1+a2;
    a1 = a2;
    a2 = t;    //(a1,a2)→(a2,a1+a2)
}

//-----④-----//
for(int i=1; i<=100; i++){
    if(i%3!=0){                //3の倍数でない
        if((i%10)!=3){        //1の位
            if(((i/10)%10)!=3){ //10の位
                if(((i/100)%10)!=3){ //100の位
                    System.out.println(i);
                }
            }
        }
    }
}

//-----⑤-----//
for(int i=1; i<=1000; i++){
    int dame = 0;
    for(int j=2; j<i; j++){
        if(i%j == 0){
            dame = 1;    //2以上i未満の数で割れるものがあたらダメ
        }
    }
    if(dame == 0){
        System.out.println(i);
    }
}

```

## Part2 グラフィックス

図形の描画、アニメーション、キー操作などの扱い方と、「クラス」を勉強します。

1. 骨組みを作る
2. 図形の描画
3. 色
4. 変数を使ったアニメーション
5. クラス①
6. クラス②
7. リストクラス
8. クラスの継承
9. 練習
10. キー操作

## 1. 骨組みを作る

今までは文字と数字しか扱ってきませんでしたが、ここからは画像やアニメーションを扱っていくことになります。まずは以下のプログラムを書きましょう。ファイル名は「Part2\_01\_template」としてください。

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4 import java.util.*;
5
6 public class Part2_01_template extends Applet implements Runnable{
7     int w = 500;           //画面の横幅
8     int h = 500;           //画面の縦幅
9     int step;               //プログラムが始まってから何ステップたったかを記録
10
11     public void init(){    //最初に一回実行されるメソッド
12         setSize(w,h);      //画面の大きさを、横w、縦hにする
13         step = 0;           //最初にstepを0にしておく。
14         Thread th = new Thread(this); //このプログラムを実行する準備
15         th.start();         //run()メソッドを開始
16     }
17
18     public void keisan(){  //1ステップごとに実行される計算
19         step++;            //stepに1を加算
20     }
21
22     public void paint(Graphics g){ //1ステップごとに書かれるグラフィック
23         g.drawString("step = " + step, 0, h-10); //現在のステップ数を表示
24     }
25
26     public void run(){
27         try{
28             for(;;){        //無限ループ
29                 repaint();  //画面を一回白で消して、paintを再実行する
30                 Thread.sleep(10); //10ミリ秒(0.01秒)停止する
31                 keisan();    //keisanメソッドを実行する
32             }
33         }catch(Exception err){
34         }
35     }
36 }
37
38 }
39
40

```

実行したら白い画面が出てきて、左下の step がどんどん増えていくのが表示されると思います。これからのプログラムは、この template をコピー＆貼りつけて作っていくことになります。

4つのメソッドの説明をします。

init: 一番最初に1回だけ実行されるメソッド。プログラムの準備をする担当。

keisan: 1ステップごとに実行する処理。

paint: 画像を描画するメソッド。1ステップごとに書きなおされる。

run: keisan と paint を呼び出しているメソッド。ここでは0.01秒に1回ずつ呼び出し続けている。

## 2. 図形の描画

ここでは図形(直線、円、四角形、文字)の描画方法を勉強します。

新しいファイル「Part2\_02\_zukei」を作りましょう。作ったら template をコピーして貼りつけてください。  
クラス名のところを次のように新しい名前に書き換えましょう。

```
public class Part2_02_zukei extends Applet implements Runnable{
```

次に、paint の中身に次のように書き加えてください。

```
public void paint(Graphics g){
    g.drawLine(10, 40, 10, 160); // (10,40)から(10,160)への直線
    g.drawRect(10, 10, 80, 30); // □ (左上が(10,10)、幅80、高さ30)
    g.fillOval(15, 15, 20, 20); // ● (左上が(15,15)、幅20、高さ20)
    g.drawOval(40, 15, 20, 20); // ○
    g.drawOval(65, 15, 20, 20); // ○
    g.drawString("信号機", 50, 180);
    g.drawString("step = " + step, 0, h-10); //現在のステップ数を表示
}
```

このプログラムの説明をします。

#### ①座標について

ウィンドウ上の位置は、二つの数の組(x,y)で表します。xは左端からの距離、yは上端からの距離です。

このプログラムでは7, 8行目くらいでwとhに500をいれているので、ウィンドウの大きさは縦横ともに500です。

#### ②直線の描画

```
g.drawLine(x1, y1, x2, y2);
```

(x1,y1)から(x2,y2)への直線を引きます。

#### ③図形(だ円と四角形)の描画

```
g.drawRect(x, y, w, h);
```

(x,y)を左上の角とする、幅w、高さhの長方形

```
g.drawOval(x, y, w, h);
```

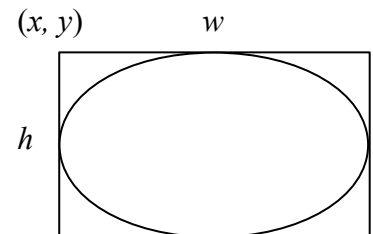
↑の長方形に内接するだ円

```
g.fillRect(x, y, w, h);
```

四角形を塗りつぶしたもの

```
g.fillOval(x, y, w, h);
```

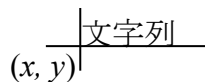
だ円を塗りつぶしたもの



#### ④文字の描画

```
g.drawString(文字列, x, y);
```

(x,y)を左下の角とする文字列を表示します。





図形に色を付けてみましょう。

先ほどと同じように「Part2\_03\_color」を作って、paint メソッドは次のようにしてください。

```
public void paint(Graphics g){
    g.setColor(Color.blue);
    g.fillRect(0, 0, 160, 320);
    g.setColor(Color.white);
    g.fillRect(160, 0, 160, 320);
    g.setColor(Color.red);
    g.fillRect(320, 0, 160, 320);

    g.setColor(Color.black);
    g.drawString("フランス", 220, 360);
    g.drawString("step = " + step, 0, h-10); //現在のステップ数を表示
}
```

#### ①色の付け方

setColor 文は、再び setColor が実行されるまで全ての図形が ( ) 内の色で描かれることになります。

g. setColor (Color. blue); (青色で描画するようにする)

#### ②色の種類

黒	Color. black
白	Color. white
赤	Color. red
緑	Color. green
青	Color. blue
黄	Color. yellow
水	Color. cyan
桃	Color. magenta

・ドイツの国旗も書いてみましょう。



## 4. 変数を使ったアニメーション

1 ステップごとに変数の値を変えて、その変数を基に図形を描画することで、図形を動かすことができます。今までと同じように「Part2\_04\_animation」を作って、各メソッドや変数を次のように書いてください。

```

5 public class Part2_04_animation extends Applet implements Runnable{
6     int w = 500;
7     int h = 500;
8     int step;
9
10    int x;
11    int y;                //位置
12    int vx;
13    int vy;                //速度
14
15    public void init(){
16        setSize(w,h);
17        step = 0;
18        x = 50;
19        y = 50;
20        vx = 10;
21        vy = 7;            //各変数を初期化する
22        Thread th = new Thread(this);
23        th.start();
24    }
25
26    public void keisan(){
27        x += vx;            //x方向に移動
28        y += vy;            //y方向に移動
29        step++;
30    }
31
32    public void paint(Graphics g){
33        g.fillOval(x-5,y-5,10,10);    //(x,y)を中心とする直径10の●を書く
34        g.drawString("step = " + step, 0, h-10);
35    }
36
37    public void run(){
38        try{
39            for(;;){
40                repaint();
41                Thread.sleep(10);
42                keisan();
43            }
44        }catch(Exception err){
45        }
46    }
47
48 }
49 }
```

●が右下の方に飛んでいくアニメーションが表示されると思います。

- ・ init 中の x,y,vx,vy に代入している値を変えて、アニメーションがどう変化するか実験してみましょう
- ・ ●が壁にあたったら反射するようにしてください。

ヒント: 横方向の反射は、vx に-1 を掛ければ、縦方向は vy に-1 を掛ければできます。

## 5. クラス (1)

## ①クラスとは

クラスとは、「いくつかのデータと、そのデータを操作するメソッドをまとめたもの」です。

身長、体重のデータと、BMI(肥満度)を求めるメソッドを組み合わせた Person クラスを作ってみましょう。

今までと同じように「Part2\_05\_class1」を作ってください。template もコピーしてください。

## ②クラスの定義

クラスの定義は import...と書かれている行と、public class Part2\_05\_class1 ...と書かれている行の間で行ないます。実は、今まで何度も書いてきた public class...というのもクラスの定義だったのです。クラスの中でクラスを定義することはできないので、public class...の前の部分で新たなクラスの定義を行ないます。また、次の章で実際にやりますが、別のファイルで定義されたクラスも使うことができます。

```
class Person{
    double shinchou;
    double taijuu;

    double keisanBMI(){
        return taijuu/(shinchou*shinchou);
    }
}
```

このように書いてください。見てわかるとおり、Person クラスは、double 型の変数 shinchou,taijuu と、double を返すメソッド keisanBMIを持っています。

## ③クラスの宣言と実体化

変数と同様、Person a; のようにまず宣言しないといけません。

変数と違うのは、この後さらに、a = new Person(); と書かなければならないということです。この違いは、変数には値そのものが記録されるのに対し、この「a」には a の中身である shinchou や taijuu といった変数の「場所」が記録されるために生じます。宣言した段階では「a」には何も無い場所が入っていて、a = new Person();と書くことで始めて、shinchou や taijuu が新たに作られ、a がその場所を指すようになります。

int step;の次の行で、次のように Person 型の a とbを宣言しましょう。

```
Person a, b;
```

## ④クラスの使い方

a の中身を操作するには、「.」を使います。a.shinchou = 1.655; のように書けば a の shinchou に値 1.655 を代入できます。a.shinchou と b.shinchou は違うものを指すことに注意しましょう。a のメソッドを呼び出す時も同様で、a.keisanBMI()と書きます。a.keisanBMI()と b.keisanBMI()は、それぞれ身長、体重が異なるので違う結果を返します。init メソッドと paint メソッドを次のように書きましょう。

```
public void init(){
    setSize(w,h);
    step = 0;

    a = new Person();    //実体化
    b = new Person();
    a.shinchou = 1.655;   //aの身長 (m単位)
    a.taijuu = 45.2;      //aの体重 (kg単位)
    b.shinchou = 1.785;
    b.taijuu = 80.5;

    Thread th = new Thread(this);
    th.start();
}

public void paint(Graphics g){
    g.drawString("aのBMIは" + a.keisanBMI(), 100, 100);
    g.drawString("bのBMIは" + b.keisanBMI(), 100, 130);
    g.drawString("step = " + step, 0, h-10); //現在のステップ数を表示
}
```

## 6. クラス (2)

## ①ファイルを分ける

クラスの定義はしばしば長くなるので、読みやすくするためにファイルを分けます。クラスは別のファイルからでも使えるからです。

今までと同じように、「Part2\_06\_class2」を作って、template をコピーしてください。

この他に、もうひとつ「Point」という名前のファイルを作ってください。こっちには template はコピーしないでください。Part2\_06\_class2 の側から、Point クラスを使ってみることにします。

## ②コンストラクタ

実体化する際に行うメソッドをクラスに書くと便利です。これはコンストラクタといい、実体化の際についでに変数に値を代入したりするのに使われます。

このメソッドは、メソッド名がクラス名と同じで、戻り値はなし、引数はあっても大丈夫です。

Point ファイルを次のように編集してください。

```

1 import java.awt.*;
2
3 public class Point {
4     int x; //座標
5     int y; //座標
6     int r; //半径
7
8     Point(int x1, int y1, int r1){ //コンストラクタ
9         x=x1; //x,y,rそれぞれに引数でとった値を入れる
10        y=y1;
11        r=r1;
12    }
13
14    void paint(Graphics g){
15        g.fillOval(x-r, y-r, 2*r, 2*r); //(x,y)を中心とする半径rの円
16    }
17 }
18

```

コンストラクタの引数 x1,y1,r1 に渡す値は、new 文の中で、new Point(50, 100, 5);のように書きます。

コンストラクタの他にもう一つ、自分自身を描画するメソッド paint を書いてみました。これで、Part2\_06\_class2 の方に描画処理の詳細を書かなくてもよくなります。

Part2\_06\_class2 は次のように書いてください。スペースの関係上途中で途切れていますが、後は template と同じ run メソッドと、カッコを閉じるだけです。

```

5 public class Part2_06_class2 extends Applet implements Runnable{
6     int w = 500;
7     int h = 500;
8     int step = 0;
9     Point p1,p2; //宣言
10
11     public void init(){
12         p1 = new Point(50, 100, 5); //コンストラクタを使った実体化
13         p2 = new Point(100, 50, 10); //(x,y,r)を(100,50,10)に初期化する
14         setSize(w,h);
15         Thread th = new Thread(this);
16         th.start();
17     }
18
19     public void keisan(){
20         p1.x++; //p1は右方向に
21         p2.y++; //p2は下方向に動かしてみる
22     }
23
24     public void paint(Graphics g){
25         p1.paint(g); //Pointクラスに書いたpaintメソッドを使ってみる
26         p2.paint(g);
27     }
28 }

```

## 7. クラス (3) LinkedList

先程のプログラムでは、Point 型の実体(これを**インスタンス**といいます)を p1 と p2 の 2 つ作りましたが、これをもっとたくさん作りたい時や、あるいは実行しながら作ったり消したりできるようにしたい時、LinkedList というものを使うと便利です。LinkedList のイメージとしては、いくつかのインスタンスを横に並べた物と思ってください。それにインスタンスを追加したり、あるいは削除したり、検索したりできます。

LinkedList はそれ自体がクラスです。ですから、宣言と実体化が必要になります。また、いくつかのメソッドを持っていて、インスタンスを追加する、削除するといった操作ができます。LinkedList に格納したインスタンスの情報を得るには、Iterator というこれまた特別なクラスを使います。この使い方はちょっと複雑なので、具体例を見ながらとりあえず使えるようになります。

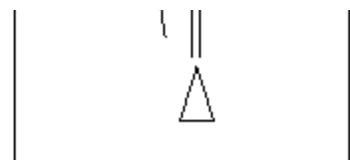
「Part2\_07\_linklist」を作って、template をコピーしてください。ここでは LinkedList への要素の追加と、要素の取り出し方を学びます。

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4 import java.util.*;
5
6 public class Part2_07_linklist extends Applet implements Runnable{
7     int w = 500;
8     int h = 500;
9     int step = 0;
10    LinkedList<Point> points; //Point型を格納するLinkedListの宣言
11
12    public void init(){
13        points = new LinkedList<Point>(); //LinkedListの実体化
14        setSize(w,h);
15        Thread th = new Thread(this);
16        th.start();
17    }
18
19    public void keisan(){
20        if(step % 10 == 0){
21            Point tmp = new Point((int)(Math.random()*w), (int)(Math.random()*h), 10);
22            //新しいPoint型のインスタンスを作って、
23            //コンストラクタでウィンドウ内のランダムな位置、半径は10にセット
24            points.add(tmp); //LinkedListにこのインスタンスを追加
25        }
26        step++;
27    }
28
29    public void paint(Graphics g){
30        Iterator<Point> it = points.iterator(); //pointsを走査するイテレータitを用意
31        for( ; it.hasNext(); ){ //itが指すインスタンスの次のインスタンスがあるかぎり繰り返す
32            Point p = it.next(); //pにitが指すインスタンスを入れて、itは次のインスタンスに移る
33            p.paint(g);
34        } //このfor文で、LinkedListの中身が全て走査される
35    }
36
37    public void run(){
38        try{
39            for(;;){
40                repaint();
41                Thread.sleep(50); //待機時間をちょっと長くしよう
42                keisan();
43            }
44        }catch(Exception err){
45        }
46    }
47
48 }
49 }
50

```

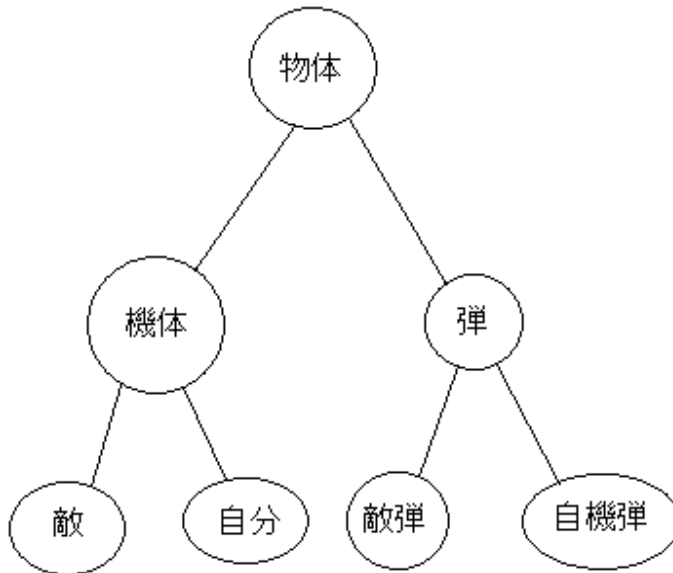
## 8. クラス (4) クラスの継承



クラスを多用してプログラミングしていくと、似たような変数やメソッドをもつクラスをいくつも作るような場面に遭遇します。右図みたいなシューティングゲームを考えてみましょう。

画面上には「敵」「自分」「敵弾」「自機弾」があります。それぞれ別のクラスを作って操作したり表示したりするのが良いでしょう。しかし、「敵」と「自分」は例えば HP とか攻撃とか似た要素を持てますし、「敵弾」と「自機弾」も速度とか命中とか似た要素を持っています。さらに、これら4つのオブジェクトは全て、画面上の座標という共通要素があります。

このように、上位のクラスと下位のクラスを作ることで、同じものを何度も書く手間が省け、改造しやすくなったり、



ミスも少なくなります。下位のクラスは、継承した上位のクラスに含まれる変数、メソッドを全て使うことができます。上位のクラスを引き継いで下位のクラスを定義する際は、「extends」というキーワードを使います。先程の Point を継承したクラス「MovingPoint」を作り、次のように書いてみましょう。

```

1
2 public class MovingPoint extends Point{
3     static int windowW; //ウィンドウの幅。
4     static int windowH; //ウィンドウの高さ
5     //☆staticとつけると、全てのインスタンスに対してこの値は共通になる。
6     int vx; //速度 (x方向)
7     int vy; //速度 (y方向)
8
9     MovingPoint(){ //コンストラクタ
10         x = (int)(Math.random()*windowW);
11         y = (int)(Math.random()*windowH); //画面内のランダムな位置に配置
12         r = 5; //半径は5
13         vx = (int)(Math.random()*10) - 5;
14         vy = (int)(Math.random()*10) - 5; //速度もランダムに
15     }
16
17     boolean move(){ //移動を行うメソッド。
18         x += vx;
19         y += vy;
20         if(x < 0 || x > windowW) return true; //画面端に達したらtrueを返す
21         if(y < 0 || y > windowH) return true;
22         return false; //画面内ならfalseを返す
23     }
24 }
25
26

```

続いて、本体のプログラムを書きましょう。「Part2\_08\_extends」を作ってください。

画面内を●が飛び回るようなアニメーションを作ってみます。ここまでくればシューティングゲームまであと一歩です。

ここでも再び LinkedList を使います。画面外に出たインスタンスを消去するときに、remove メソッドを使います。その使い方を身につけましょう。

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4 import java.util.*;
5
6 public class Part2_08_extends extends Applet implements Runnable{
7     int w = 500;
8     int h = 500;
9     int step = 0;
10    LinkedList<MovingPoint> points;           //今度はMovingPoint型のLinkedList
11
12    public void init(){
13        points = new LinkedList<MovingPoint>();
14        MovingPoint.windowW = w;             //MovingPointクラス共通の変数に
15        MovingPoint.windowH = h;             //画面の大きさを記録する
16        setSize(w,h);
17        Thread th = new Thread(this);
18        th.start();
19    }
20
21    public void keisan(){
22        if(step % 10 == 0){                   //10ステップに一回実行する
23            points.add(new MovingPoint());    //LinkedListに新しいインスタンスを追加
24        }
25
26        Iterator<MovingPoint> it = points.iterator();
27        for( ; it.hasNext(); ){
28            MovingPoint p = it.next();
29            boolean del = p.move();           //移動の処理。画面外に出たらtrueが帰る
30            if(del){                          //画面外に出ていたら
31                it.remove();                 //LinkedListから削除する
32            }
33        }
34        step++;
35    }
36
37    public void paint(Graphics g){
38        Iterator<MovingPoint> it = points.iterator();
39        for( ; it.hasNext(); ){
40            MovingPoint p = it.next();
41            p.paint(g);
42        }
43    }
44
45    public void run(){
46        try{
47            for(;;){
48                repaint();
49                Thread.sleep(10);
50                keisan();
51            }
52        }catch(Exception err){
53        }
54    }
55
56 }
57 }
58

```

## 9. 練習

- (1) 市松模様を書いてみましょう（クラスを使わなくてできます）
- (2) MovingPoint を改造して、Part2\_08\_extends で●が画面端で跳ね返るようにしてみましょう

- (3) MovingPoint を改造して、●同士が反発するようにしてみましょう（難しいです）
- A) ●同士の距離を測るメソッドを作りましょう
  - B) 距離に応じてお互いの  $v_x$ ,  $v_y$  を変化させましょう
  - C) 距離が近いほど強く反発するようにしてください



## 10.キー操作

これから上下左右キーで自機を操作するゲームを作っていきます。まずは、キーの入力をどうやって受け取るかを勉強しましょう。

## (1)アプレット本体

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4 import java.util.*;
5
6
7 public class Part2_10_key extends Applet implements Runnable, KeyListener{
8     int w = 500;
9     int h = 500;
10    int step;
11    Keydata key;           //現在どのキーが押されているかを記録するクラス
12    KeyPoint p;           //キー操作によって動く●のクラス
13
14    public void init(){
15        key = new Keydata();
16        p = new KeyPoint(50, 50, 5);
17        addKeyListener(this);           //キーの入力をうけとれるようにする
18        step = 0;
19        setSize(w,h);
20        Thread th = new Thread(this);
21        th.start();
22    }
23
24    public void keisan(){
25        p.move(key);           //現在どのキーが押されているかの情報を基に動かす
26        step++;
27    }
28
29    public void paint(Graphics g){
30        p.paint(g);
31        g.drawString("step = " + step, 0, h-10);
32    }
33
34    public void run(){
35        try{
36            for(;;){
37                repaint();
38                Thread.sleep(10);
39                keisan();
40            }
41        }catch(Exception err){
42        }
43    }
44
45
46
47    public void keyPressed(KeyEvent e) { //キーが押されたときの処理
48        key.keyPressed(e);           //Keydataクラスのメソッドに横流しする
49    }
50
51    public void keyReleased(KeyEvent e) { //キーが離されたときの処理
52        key.keyReleased(e);           //Keydataクラスのメソッドに横流しする
53    }
54
55    public void keyTyped(KeyEvent e) { //キーが押し離されたときの処理（今回はなし）
56    }
57
58 }

```

赤い四角で囲ってあるところが、今までと違うところです。

次に、現在どのキーが押されているかを記録するクラス `Keydata` と、その情報を基に動く `KeyPoint` クラスを作りましょう。

## (2)Keydata クラス

Keydata クラスには、現在どのキーが押されているか記録します。キーが押されたとき、そのキーに対応する変数を true にして、離されたときに false にします。

```

1 import java.awt.event.*;
2
3 public class Keydata {
4     boolean UP = false; // ↑ キーが押されている
5     boolean DOWN = false; // ↓
6     boolean RIGHT = false; // →
7     boolean LEFT = false; // ←
8
9     public Keydata() {}
10
11     public void keyPressed(KeyEvent e) { // キーが押されたとき
12         if (e.getKeyCode() == KeyEvent.VK_UP) UP = true;
13         else if (e.getKeyCode() == KeyEvent.VK_DOWN) DOWN = true;
14         else if (e.getKeyCode() == KeyEvent.VK_RIGHT) RIGHT = true;
15         else if (e.getKeyCode() == KeyEvent.VK_LEFT) LEFT = true;
16     }
17
18     public void keyReleased(KeyEvent e) { // キーが離されたとき
19         if (e.getKeyCode() == KeyEvent.VK_UP) UP = false;
20         else if (e.getKeyCode() == KeyEvent.VK_DOWN) DOWN = false;
21         else if (e.getKeyCode() == KeyEvent.VK_RIGHT) RIGHT = false;
22         else if (e.getKeyCode() == KeyEvent.VK_LEFT) LEFT = false;
23     }
24 }
25
26

```

## (3)KeyPoint クラス

押されているキーの情報を基に自機を動かす move メソッドを持った、Point クラスの下位クラスです。

```

1
2 public class KeyPoint extends Point{
3
4     KeyPoint(int x1, int y1, int r1){
5         x=x1;
6         y=y1;
7         r=r1;
8     }
9     void move(Keydata key){
10         if(key.UP) y-=5;
11         if(key.DOWN) y+=5;
12         if(key.LEFT) x-=5;
13         if(key.RIGHT) x+=5;
14     }
15 }
16
17

```

### Part3 シューティングゲームを作ろう

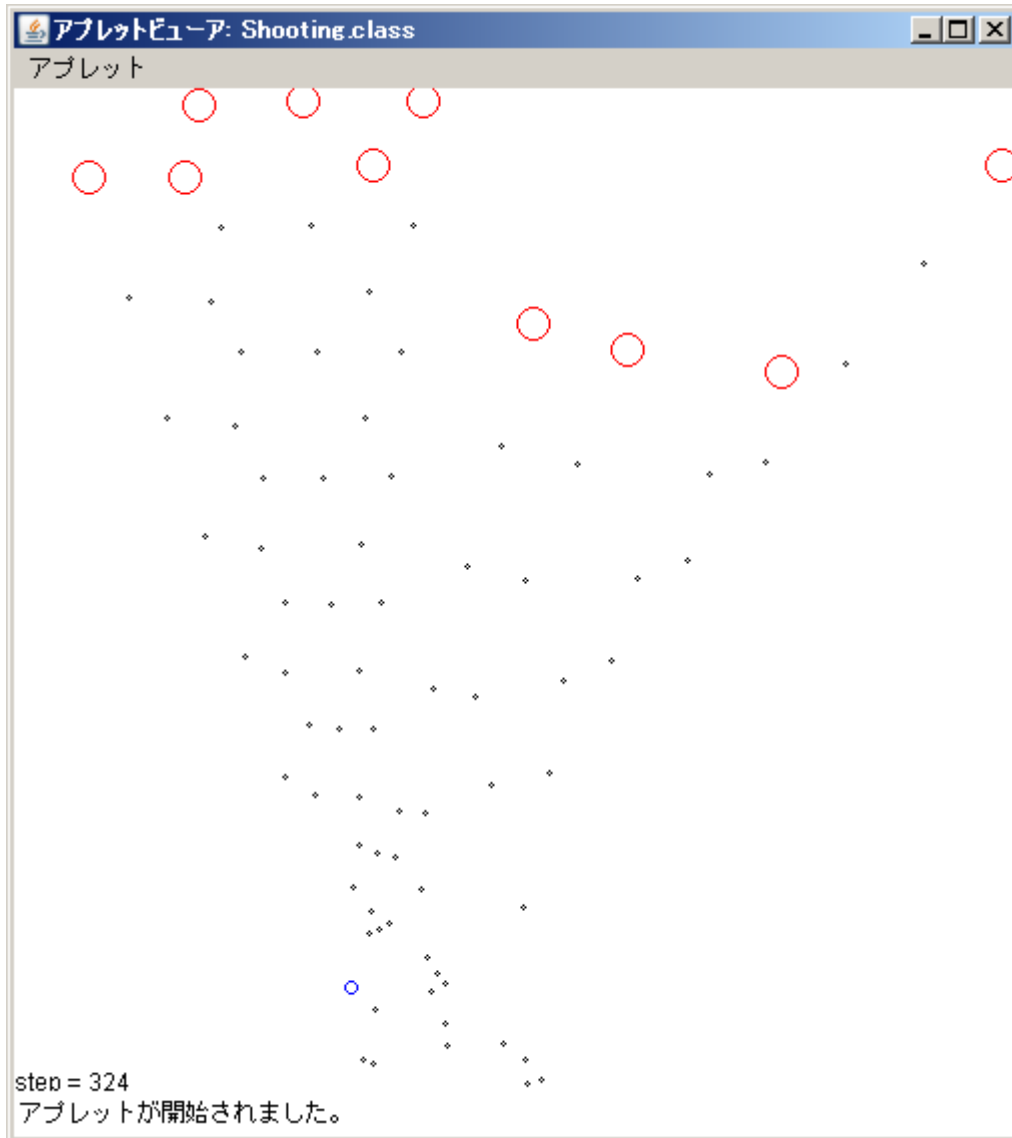
まずは単純なものを作って、それを改良していきましょう。

1. シューティングゲームの説明
2. 骨組みを作る
3. 画面上の物体 : Item クラス
4. 弾 : Bullet クラス
5. 機体 : Kitai クラス
6. 敵 : Enemy クラス
7. キー操作 : Keydata クラス
8. 自機 : Player クラス
9. ゲーム本体 : Shooting クラス
10. 改良しよう

## 1. シューティングゲームの説明

シューティングゲームとは弾を撃って敵を倒していくタイプのゲームの事を言います。ここでは、オーソドックスな「縦スクロールシューティングゲーム」を作しましょう。

今日の目標は、下の画面のような非常にシンプルなシューティングゲームを完成させることです。



青が自機、赤が敵機で、弾を撃ち合います。自機は上下左右キーで移動し、Zキーで弾を撃ちます。どちらも一定数以上ダメージを受けると死にます。

これをどうやってプログラムにしていけばよいでしょうか。まず、「敵」や「弾」といったオブジェクトが必要です。これらはクラスで定義するのが楽でしょう。弾と機体の命中判定も必要ですし、いつ攻撃するか、どの方向に弾を撃つかといった命令も必要です。自機の操作には、先ほどやったキー操作の命令が必要でしょうし、画面外に出た弾を消す、といった昨日学んだ LinkedList の操作も必要です。

さて、ここから具体的にプログラムを書いていきます。今まで書いてきたものに比べると数倍長く、かつ、なかなか動くようにならないので大変だと思いますが、どうか頑張ってゲームを完成させてみてください。

プログラムのこの部分がわからない！というときは気軽に声をかけてください。分からないものを書くのはよくありません。

## 2. 骨組みを作る

シューティングゲームを作るにあたって、まずは「Item」、「Bullet」、「Kitai」、「Player」、「Enemy」、「Keydata」、「Shooting」の7つのクラスを作ります。

それぞれのクラスがどんな働きをしていて、どんな変数を持っていて、どんなメソッドがあるのかを見渡すために、ここでは各クラスの外枠だけを書いてみましょう。中身はその後に順番に一つずつ作っていきます。

ちゃんと動くものが出来上がるまでに、かなり時間がかかると思いますが、辛抱してください。

今回は、プログラムを書くときは、//とその後の日本語も一緒に写してください。この//は「コメント」といって、コンピュータは無視しますが、人間が読んで分かりやすくするために使われるものです。このくらい大きなプログラムになると、コメントをつけておかないと後で見たときに意味不明になってしまいます。

### Item クラス

画面上に表示するオブジェクトは、全てこの Item クラスを継承させることにします。画面の情報、位置、大きさ、状態、色といった変数を持ち、自分自身を描画するメソッド paint を持たせます。

敵、自機の生死は、ここの state 変数で管理します。これが 0 のとき死亡、1 のとき生存とします。

```

1 import java.awt.*;
2
3 public class Item { //画面上に存在する物体を表すクラス
4     static double windowW; //画面の横幅
5     static double windowH; //画面の縦幅
6     double x;
7     double y; //位置
8     double r; //半径
9     int state; //状態 0:dead 1:alive
10    Color c; //色
11
12    void paint(Graphics g){ //このItemを描画する
13    }
14 }
```

### Bullet クラス

```

1 import java.awt.*;
2
3 public class Bullet extends Item{ //弾を表すクラス
4     double vx;
5     double vy; //速度
6
7     //コンストラクタ
8     public Bullet() {
9     }
10
11    //移動処理
12    public void action() {
13    }
14
15    //画面端判定
16    public boolean isOut() {
17        return false;
18    }
19
20    //命中判定
21    public boolean hit(Item target) {
22        return false;
23    }
24
25 }
```

Item クラスを継承して、弾丸を表す Bullet クラスを定義します。Item クラスの変数に加えて、速度を表す変数が付け加わり、移動、画面端、命中判定といったメソッドを定義します。

## 11. Kitai クラス

今度は機体を表すクラスを定義します。これは Item クラスを継承しており、一方で自機、敵機を表すクラスに継承されるクラスになります。ヒットポイントと攻撃に関する要素が付け加わります。

```

1 import java.awt.*;
2
3 public class Kitai extends Item{ //機体を表すクラス
4     int hp; //体力
5     int coolingtime; //攻撃間隔
6
7     int cooling; //後何ステップで攻撃できるか
8
9     //コンストラクタ
10    Kitai(){
11    }
12
13    //ダメージ時の処理
14    public void damage() {
15    }
16}

```

## (4) Player クラス

Kitai クラスを継承し、自機をあらわす Player クラスを定義します。移動、攻撃の処理を表す action メソッドは、

```

1 import java.awt.*;
2 import java.util.*;
3
4 public class Player extends Kitai{
5
6     //コンストラクタ
7     Player() {
8     }
9
10    //移動・攻撃処理
11    public void action(Keydata key, LinkedList<Bullet> bullets) {
12    }
13
14}

```

現在押されているキーが記録されている Keydata クラスのインスタンスと、攻撃の際に発射した弾を追加する先である、Bullet 型のリストを引数に取ります。

## (5) Enemy クラス

```

1 import java.awt.*;
2 import java.util.*;
3
4 public class Enemy extends Kitai{
5
6     //コンストラクタ
7     Enemy() {
8     }
9
10    //攻撃処理
11    public void action(Player player, LinkedList<Bullet> bullets) {
12    }
13
14}

```

Player クラスと同様ですが、攻撃処理の際に、自機の情報を参考に攻撃を行ないます。

## (6) Keydata クラス

Part2 で作成した Keydata をコピーしてきましょう。後で攻撃キー(Z キー)に関する処理を加えます。

## (7) Shooting クラス

このクラスがゲームの本体になります。長いですが頑張って！

```

1 import java.awt.*;
2 import java.awt.event.*;
3 import java.applet.*;
4 import java.util.*;
5
6 public class Shooting extends Applet implements Runnable,KeyListener{
7     int w = 500;
8     int h = 500;
9     int step = 0;
10
11     Keydata key;
12
13     Player player;
14     LinkedList<Enemy> enemies; //自機
15     LinkedList<Bullet> bulletsE; //敵機のリスト
16     LinkedList<Bullet> bulletsP; //敵弾のリスト
17     //自弾のリスト
18
19     public void init(){
20         setSize(w,h);
21         addKeyListener(this);
22         key = new Keydata();
23
24         setGame();
25
26         Thread th = new Thread(this);
27         th.start();
28     }
29
30     public void setGame(){
31         //初期化处理
32     }
33
34     public void nextStep(){
35         //敵の行動
36         //自機の行動
37         //弾の移動
38         //当たり判定
39         //自機
40         //敵機
41         //画面外に出た弾を消す
42         step++;
43     }
44
45     public void paint(Graphics g){
46         //自機描画
47         //敵機描画
48         //敵弾描画
49         //自弾描画
50         g.drawString("step = " + step, 0, h);
51     }
52
53     public void run(){
54         try{
55             for(;;){
56                 nextStep();
57                 repaint();
58                 Thread.sleep(20);
59             }
60         }catch(Exception err){
61         }
62     }
63
64     public void keyPressed(KeyEvent e) {
65         key.keyPressed(e);
66     }
67
68     public void keyReleased(KeyEvent e) {
69         key.keyReleased(e);
70     }
71
72     public void keyTyped(KeyEvent e) {
73     }
74 }

```

### 3. Item クラス

ここまで書けたら、まずはどこにもエラーがない(赤線がない)ことを確認して下さい。Shooting クラスを実行すると、白い画面に、左下に step 数が表示されるはずです。

さて、ここからは、各クラスの今まで書いていなかった部分を埋めていきます。各クラスの細かい説明も一緒に書いていきますので、読んでください。

Item クラスは、描画用のメソッド paint がまだ書かれていませんでした。ここを、次のように書きましょう。

```
void paint(Graphics g){ //このItemを描画する
    if(state == 1){
        g.setColor(c);
        g.drawOval((int)(x-r), (int)(y-r), (int)r*2, (int)r*2);
    }
}
```

これで、Item クラスと Item を継承したクラスで paint を実行すると、(x,y)を中心とする半径 r の円を、色 c で書きます。今の段階では、敵、自分、弾は全て●で表します。それぞれ半径と色を変えて区別します。



## 4. Bullet クラス

4つのメソッドの中身は次のように書いてください。

```
//コンストラクタ
public Bullet(double x0, double y0, double vx0, double vy0, int r0, Color c0, int state0) {
    x = x0;
    y = y0;
    vx = vx0;
    vy = vy0;
    r = r0;
    c = c0;
    state = state0;
}

//移動処理
public void action() {
    x += vx;
    y += vy;
}

//画面端判定
public boolean isOut() {
    if (x < 0 || x > windowW || y < 0 || y > windowH) return true;
    else return false;
}

//命中判定
public boolean hit(Item target) {
    if (target.state == 1) {
        if (Math.hypot(x - target.x, y - target.y) < this.r + target.r) return true;
    }
    return false;
}
```

コンストラクタは引数の数が多くてややこしいですが、やっていることは Bullet クラスが持っている全ての変数に引数で指定した値を入れているだけです。

画面端判定は、画面の外に出たときに true を返します。

命中判定は、判定対象となる Item を継承した型のインスタンスとの距離を調べ (Math.hypot())、両者の半径の和よりもそのインスタンスとの距離が近かったら、命中したとみなして true を返します。

## 5. Kitai クラス

次は機体を表すクラスです。  
 コンストラクタとダメージ時の処理の中身を書きましょう。

```
//コンストラクタ
Kitai(double x0, double y0, int r0, Color c0, int hp0, int ctime0, int state0, int cooling0){
    x = x0;
    y = y0;
    r = r0;
    c = c0;
    hp = hp0;
    coolingtime = ctime0;
    state = state0;
    cooling = cooling0;
}

//ダメージ時の処理
public void damage() {
    hp--;
    if(hp==0){                //死亡時の処理
        state = 0;
    }
}
```

ダメージ時の処理は、hp を 1 減らして、もし 0 になったら状態を表す変数 state を 0(死亡状態)にします。この処理は自機でも敵機でも共通なので、ここに書きます。

## 6. Enemy クラス

//コンストラクタ

```
Enemy(double x0, double y0, int r0, Color c0, int hp0, int ctime0, int state0, int cooling0) {
    super(x0, y0, r0, c0, hp0, ctime0, state0, cooling0);
}
```

//攻撃処理

```
public void action(Player player, LinkedList<Bullet> bullets) {
    if(state==1){
        if(cooling == 0){
            double dx = player.x - x;
            double dy = player.y - y;
            double tmp = Math.hypot(dx, dy);
            Bullet b = new Bullet(x, y, 3.0*dx/tmp, 3.0*dy/tmp, 1, Color.BLACK, 1);
            bullets.add(b);
            cooling = coolingtime;
        }else{
            cooling--;
        }
    }
}
```

コンストラクタの super というのは、継承元のクラスのコンストラクタを使うという意味です。つまり、Kitai のコンストラクタが使われるので、引数にとった各パラメータを変数に代入するだけです。

攻撃処理は action メソッドで行われます。

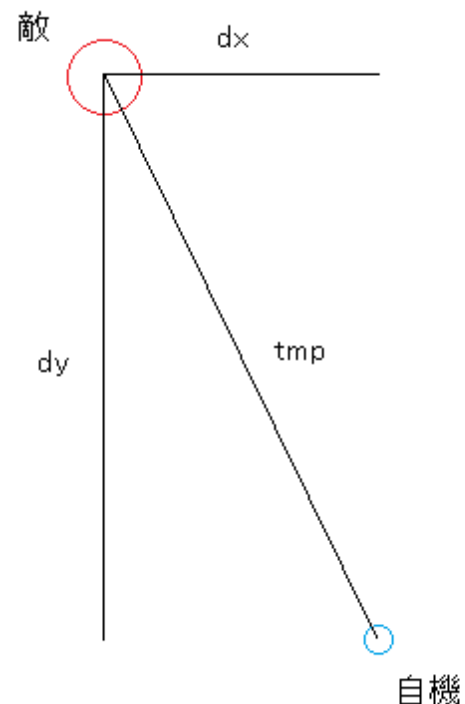
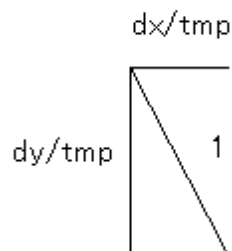
ステップごとに cooling が 1 ずつ減っていき、0 になったら攻撃して、再び cooling を変数 coolingtime に入っている値にします。これで攻撃間隔が調整できる仕組みです。

new Bullet() の中身の  $3.0 \cdot dx / tmp$ ,  $3.0 \cdot dy / tmp$  について説明します。

これは新しく生成する弾丸の速度の、x 方向と y 方向で、速度は 3.0、方向は自機を向きます。

右図のように、dx は敵から自機までの横方向の距離、dy は縦方向の距離、tmp は直線距離です。

vx,vy にそのまま dx,dy を代入すると、方向は自機を向いているものの速度がまちまちになってしまうので、自機までの距離で割ってから、適当な決まった数(ここでは 3.0)を掛けます。



## 7. Keydata クラス

```

1 import java.awt.event.*;
2
3 public class Keydata {
4     boolean UP = false;    // ↑ キーが押されている
5     boolean DOWN = false;  // ↓
6     boolean RIGHT = false; // →
7     boolean LEFT = false;  // ←
8     boolean Z = false;     // Z
9
10    public Keydata() {}
11
12    public void keyPressed(KeyEvent e) { // キーが押されたとき
13        if (e.getKeyCode() == KeyEvent.VK_UP)    UP = true;
14        else if (e.getKeyCode() == KeyEvent.VK_DOWN) DOWN = true;
15        else if (e.getKeyCode() == KeyEvent.VK_RIGHT) RIGHT = true;
16        else if (e.getKeyCode() == KeyEvent.VK_LEFT) LEFT = true;
17        else if (e.getKeyCode() == KeyEvent.VK_Z)  Z = true;
18    }
19
20    public void keyReleased(KeyEvent e) { // キーが離されたとき
21        if (e.getKeyCode() == KeyEvent.VK_UP)    UP = false;
22        else if (e.getKeyCode() == KeyEvent.VK_DOWN) DOWN = false;
23        else if (e.getKeyCode() == KeyEvent.VK_RIGHT) RIGHT = false;
24        else if (e.getKeyCode() == KeyEvent.VK_LEFT) LEFT = false;
25        else if (e.getKeyCode() == KeyEvent.VK_Z)  Z = false;
26    }
27 }

```

Part2 で作ったものに、Z の処理(8,17,25 行目)を加えただけです。Z キーは攻撃の為に使うことにします。

## 8. Player クラス

```

1| import java.awt.*;
2| import java.util.*;
3|
4| public class Player extends Kitai{
5|
6|     //コンストラクタ
7|     Player(double x0, double y0, int r0, Color c0, int hp0, int ctime0, int state0, int cooling0) {
8|         super(x0, y0, r0, c0, hp0, ctime0, state0, cooling0);
9|     }
10|
11|     //移動・攻撃処理
12|     public void action(Keydata key, LinkedList<Bullet> bullets) {
13|         if(state==1){
14|             if(key.UP && y > 0.0){ //画面端を超えないようにしつつ、キーに合わせて移動
15|                 y -= 2.0;
16|             }
17|             if(key.DOWN && y < windowHeight){
18|                 y += 2.0;
19|             }
20|             if(key.RIGHT && x < windowW){
21|                 x += 2.0;
22|             }
23|             if(key.LEFT && x > 0.0){
24|                 x -= 2.0;
25|             }
26|             if(key.Z){ //攻撃処理
27|                 if(cooling == 0){
28|                     Bullet b = new Bullet(x, y, 0, -5.0, 1, Color.BLACK, 1); //新しい弾の作成
29|                     bullets.add(b); //リストへの追加
30|                     cooling = coolingtime; //次の攻撃まで間を取る
31|                 }
32|             }
33|             if(cooling>0) cooling--;
34|         }
35|     }
36| }
37|

```

コンストラクタは Enemy クラスと同様です。

移動処理は、引数にとった Keydata クラスのインスタンス key をもとに行ないます。例えば↓キーが押されていて、かつ画面の一番下にいなければ、自機を y 方向に移動します。

Z が押されているときは攻撃処理を行ないます。Enemy クラスと同じように変数 cooling を使って攻撃間隔を調整します。生成する弾は、(vx,vy)=(0, -5.0)となるので、上方に速度 5.0 で向かいます。

## 9. Shooting クラス

いよいよこれで最後のクラスです。とはいえこれがゲーム本体なので一番大変なのですが…  
setGame, paint, nextStep メソッドの中身を書いていきましょう。

```
public void setGame(){
    //初期化処理
    Item.windowW = w;
    Item.windowH = h;    //ウィンドウの大きさをItemクラスのstatic変数に記録しておく
    player = new Player(w/2, h-50, 3, Color.BLUE, 4, 10, 1, 20);
    //x, y, r, 色, HP, coolingtime, state, 最初のcooling
    enemies = new LinkedList<Enemy>();
    bulletsE = new LinkedList<Bullet>();
    bulletsP = new LinkedList<Bullet>();    //Listの実体化

    for(int i=0; i<10; i++){ //敵はとりあえず10機
        Enemy e = new Enemy(w*Math.random(), h*Math.random()/3, 8, Color.RED, 3, 20, 1, 30);
        enemies.add(e);
        //x, y, r, 色, HP, coolingtime, state, 最初のcooling
    }
}
```

setGame は init メソッドの中で呼び出しているメソッドで、init と同じくゲーム開始時に使われます。ここでは、各クラスの実体化と、敵を 10 体用意します。自機は Player メソッドのコンストラクタを用いて、位置:(w/2, h-50)、半径:3、色:青、HP:4、攻撃間隔:10 ステップ…というように初期化されます。敵の位置は画面上方のランダムな場所にします。

```
public void paint(Graphics g){
    //自機描画
    player.paint(g);
    //敵機描画
    for(Iterator<Enemy> it = enemies.iterator(); it.hasNext(); ){
        Enemy e = it.next();
        e.paint(g);
    }
    //敵弾描画
    for(Iterator<Bullet> it = bulletsE.iterator(); it.hasNext(); ){
        Bullet b = it.next();
        b.paint(g);
    }
    //自弾描画
    for(Iterator<Bullet> it = bulletsP.iterator(); it.hasNext(); ){
        Bullet b = it.next();
        b.paint(g);
    }
    //ステップ数描画
    g.setColor(Color.BLACK);
    g.drawString("step = " + step, 0, h);
}
```

Player、Enemy、Bullet はすべて Item クラスを継承しているので、Item クラスのメソッドである paint が使えます。これを用いて、すべての機体と弾を描画します。

最後に nextStep メソッドです。ページをめくってください。

最後にして最大の難関、nextStep メソッドです。

Enemy、Player、Bullet クラスには、それぞれ 1 step ごとの行動をあらわす action メソッドが定義されているので、まずはそれを用いて移動や攻撃をします。

次に、全ての敵弾について自機と命中しているか、全ての自弾と敵機の組み合わせについて命中しているかの判定を行い(これは Bullet クラスのメソッド hit を使う)、命中していたらその機体の damage メソッドを呼び出して、命中した弾をリストから remove メソッドで消去します。ここで消さないと、同じ弾に何度も命中して瞬殺されてしまいます。

最後に、画面外に出た弾を remove メソッドでリストから除去します。画面外かどうかの判定は、Bullet クラスの isOut メソッドを使います。

```

45 public void nextStep(){
46     //敵の行動。Listにある敵を全て見て、それぞれEnemyクラスで定義されているact
47     for(Iterator<Enemy> it = enemies.iterator(); it.hasNext(); ){
48         Enemy e = it.next();
49         e.action(player, bulletsE);
50     }
51
52     //自機の行動
53     player.action(key, bulletsP);
54
55     //弾の移動
56     for(Iterator<Bullet> it = bulletsE.iterator(); it.hasNext(); ){ //敵弾
57         Bullet b = it.next();
58         b.action();
59     }
60     for(Iterator<Bullet> it = bulletsP.iterator(); it.hasNext(); ){ //自弾
61         Bullet b = it.next();
62         b.action();
63     }
64
65     //当たり判定
66     for(Iterator<Bullet> it = bulletsE.iterator(); it.hasNext(); ){
67         Bullet b = it.next();
68         //敵弾と自機
69         boolean hit = b.hit(player);
70         if(hit){
71             player.damage();
72             it.remove();
73         }
74     }
75     for(Iterator<Bullet> it = bulletsP.iterator(); it.hasNext(); ){
76         Bullet b = it.next();
77         //自弾と敵機
78         for(Iterator<Enemy> it2 = enemies.iterator(); it2.hasNext(); ){
79             Enemy e = it2.next();
80             boolean hit = b.hit(e);
81             if(hit){
82                 e.damage();
83                 it.remove();
84                 if(e.state == 0) it2.remove();
85                 break;
86             }
87         }
88     }
89
90     //画面外に出た弾を消す
91     for(Iterator<Bullet> it = bulletsE.iterator(); it.hasNext(); ){
92         Bullet b = it.next();
93         boolean del = b.isOut();
94         if(del){
95             it.remove();
96         }
97     }
98     for(Iterator<Bullet> it = bulletsP.iterator(); it.hasNext(); ){
99         Bullet b = it.next();
100         boolean del = b.isOut();
101         if(del){
102             it.remove();
103         }
104     }
105     step++;
106 }

```

## 10. 改良しよう

- ・勝ち、負けの判定を行おう。

敵を全部倒したら勝ち、自分の HP が 0 になったら負け。決着がついたらそれを画面に表示するようにしましょう。現在生存している敵の数は、enemies リストに入っている敵の数でわかります。「もし生存している敵の数が 0 になったら・・・」は、

```
if(enemies.size() == 0)
```

と書けます。

- ・タイトル画面

Z キーを押すまでゲームが始まらないようにしましょう。

Shooting クラスの run メソッドの、for(;;)の部分に工夫する必要があります。

- ・敵の出現パターンを作ろう

決まった時間に新たな敵が増えるといった処理を書いてみましょう。

- ・敵の攻撃パターンを変えてみよう

今は自機を目掛けて撃っているだけですが、ランダムに撃つとか規則的に撃つとかやってみましょう。

- ・いろいろな弾を作ってみよう

弾の大きさを変えてみたり、速度や、進み方を変えてみましょう。



#### Part4 シューティングゲームを改良しよう

1. 勝敗を判定し、表示する
  2. タイトル画面、ゲームオーバー画面を作る
  3. 敵の攻撃パターンを増やす
  4. 画像を表示する
  5. 敵の情報をファイルから読み込む
  6. マップの情報をファイルから読み込む
- 付録： 環境設定

## 1. 勝敗を判定し、表示する

敵が全滅したら勝利、自機の HP が 0 になったら敗北です。

まずは手始めに、自機の HP を画面に表示してみましょう。

自機の HP は `player.hp` で得ることができます。

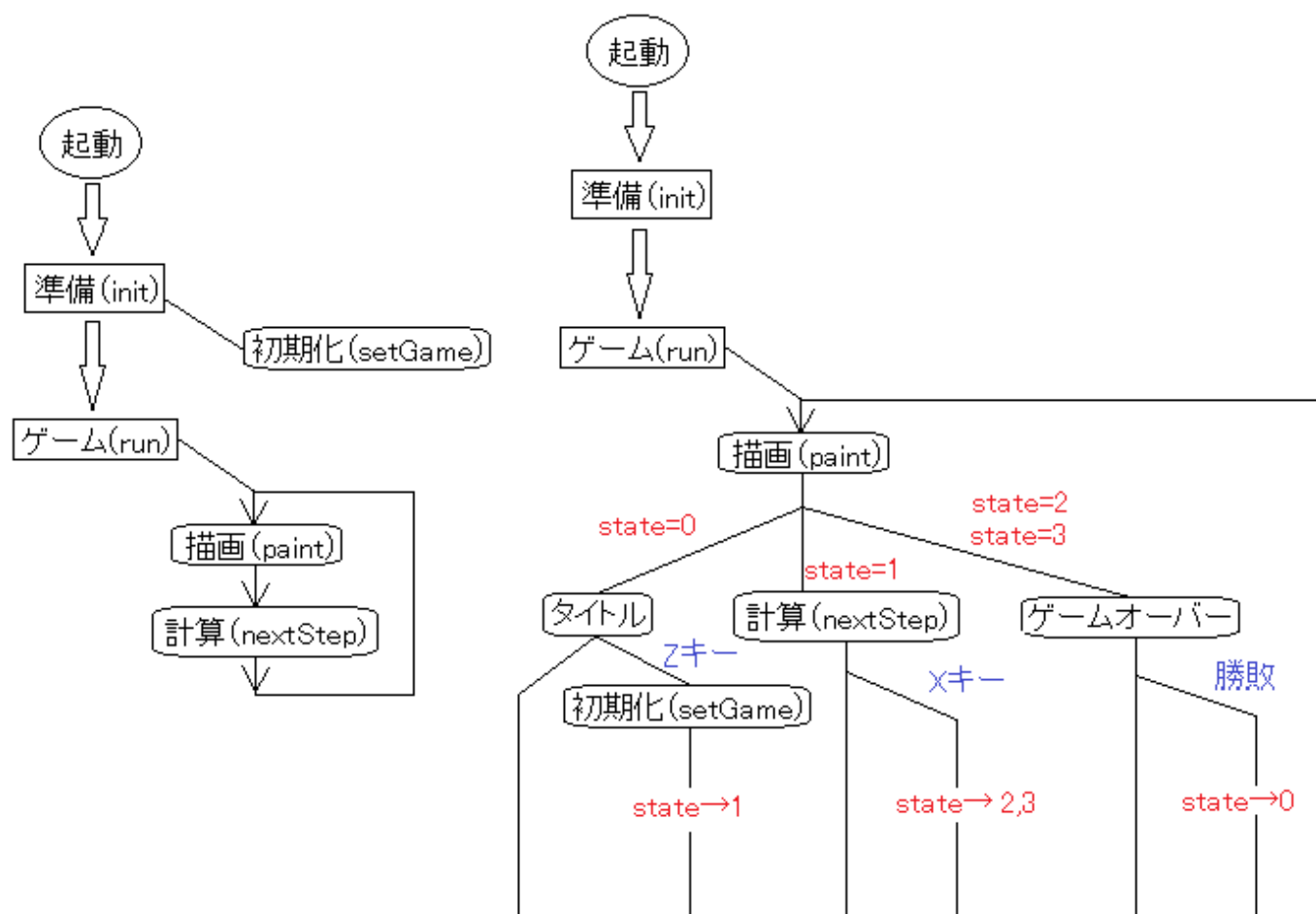
`paint` メソッドに、`g.drawString` を使って HP を表示する命令を書きましょう。

次に、自機の HP が 0 になったら負けと表示しましょう。if 文を使えば簡単にかけます。

最後に、敵の数が 0 になったら勝ちと表示しましょう。敵の数は、`enemies.size()` で得ることができます。player.hp の時とは違って、size はメソッドなので、`()` が要るので注意です。

## 2. タイトル画面、ゲームオーバー画面を作る

起動したらタイトル画面が表示されて、Zキーを押すことでゲーム開始し、勝敗がついたらゲームオーバー画面を表示します。さらに、ゲームオーバー画面からXキーを押すとタイトル画面に戻り、再びゲームを開始できるようにします。



現在のゲームの流れが上の左の図、これから作りたいのは右の図です。

現在どの画面にいるかを、state という名前の変数を使って管理します。

- 0: タイトル画面
- 1: ゲーム画面
- 2: 勝利画面
- 3: 敗北画面

### ① state の宣言

Shooting クラスの最初の方の、w や h を宣言しているところに、`int state = 0;` と書きましょう。

### ② init の変更

いままで init メソッドの中で setGame が実行されてましたが、タイトル画面から Z キーが押された段階で setGame を実行したいので、init メソッドの中の setGame を消しましょう。

## ③ paint メソッドと勝敗判定の変更

先ほど paint メソッド内に書いた勝敗判定を、nextStep メソッド内にコピーします。次に、勝ったときは state=2;、負けたときは state=3; とします。nextStep 内では g.drawString が使えないので、消します。

次に、paint メソッドを変更します。

state の値に応じて、if 文を使って場合分けし、state==0 のときタイトル画面、state==1 のとき今までと同じ描画、state==2 の時勝利画面、state==3 のとき敗北画面を表示します。それぞれの画面は drawString などを使って自分で作りましょう。

## ④ run メソッドの変更

run メソッドでは、state の値に応じて処理を切り替えます。

state==0 → key.Z が true になったら、setGame(); を実行し、state を 1 にする。

state==1 → nextStep(); を実行する。

state==2 || state==3 → key.X が true になったら、state を 0 にする。

repaint と Thread.sleep は state に関係なく実行させます。

## ⑤ Keydata の変更

今のままでは X キーを読み込んでくれないので、他のキーの文を参考に X キーも使えるようにしましょう。

参考：

paint メソッド

```
public void paint(Graphics g){
    if(state == 0){ //タイトル画面
        g.drawString("Press Z to start", w/2-30, h/2);
    }else if(state == 1){ //ゲーム画面

        (略)

    }else if(state == 2){ //勝利画面
        g.drawString("WIN", w/2-5, h/2);
        g.drawString("Press X to restart", w/2-30, h/2+20);
    }else if(state == 3){ //敗北画面
        g.drawString("LOSE", w/2-5, h/2);
        g.drawString("Press X to restart", w/2-30, h/2+20);
    }
    //ステップ数描画
    g.setColor(Color.BLACK);
    g.drawString("step = " + step, 0, h);
}
```

run メソッド

```
public void run(){
    try{
        for(;;){
            if(state == 0){
                if(key.Z == true){
                    setGame();
                    state = 1;
                }
            }else if(state == 1){
                nextStep();
            }else if(state == 2 || state == 3){
                if(key.X == true){
                    state = 0;
                }
            }
            repaint();
            Thread.sleep(20);
        }
    }catch(Exception err){
    }
}
```

### 3. 敵の攻撃パターンを増やす

敵の攻撃パターンを変えてみましょう。

#### ① AttackPattern クラスをつくる

```
import java.awt.*;
import java.util.*;

public class AttackPattern {

    public static void nerai Enemy enemy, Player player, LinkedList<Bullet> bullets, int step, int ct, double v){
        if(step % ct == 0){
            double dx = player.x - enemy.x;
            double dy = player.y - enemy.y;
            double tmp = Math.hypot(dx, dy);
            Bullet b = new Bullet(enemy.x, enemy.y, v*dx/tmp, v*dy/tmp, 1, Color.BLACK, 1);
            bullets.add(b);
        }
    }

    public static void uzumaki Enemy enemy, Player player, LinkedList<Bullet> bullets, int step, int ct, double th, double v){
        if(step % ct == 0){
            Bullet b = new Bullet(enemy.x, enemy.y, v*Math.cos((step/ct)*th), v*Math.sin((step/ct)*th), 1, Color.BLACK, 1);
            bullets.add(b);
        }
    }

    public static void housha Enemy enemy, Player player, LinkedList<Bullet> bullets, int step, int ct, int n, double v){
        if(step % ct == 0){
            for(int i=0; i<n; i++){
                Bullet b = new Bullet(enemy.x, enemy.y, v*Math.cos(i*2.0*Math.PI/n), v*Math.sin(i*2.0*Math.PI/n), 1, Color.BLACK, 1);
                bullets.add(b);
            }
        }
    }
}
```

例えば上のように AttackPattern クラスを作ります。

nerai メソッドは、敵機、自機、弾のリスト、step 数、coolingtime、弾の速度を引数として、自機狙いの弾を弾のリストに追加するメソッドです（今書いてあるものと大体同じです）。

uzumaki は渦巻状に弾を発射、housha は放射状に発射します。

Enemy クラスも書き換ええないといけません。まず、攻撃パターンを表す変数 aPattern を用意します。次に、aPattern の値に応じて AttackPattern クラスのメソッドを呼び出します。

```
1|import java.awt.*;
2|import java.util.*;
3|
4|public class Enemy extends Kitai{
5|    int aPattern = 1;
6|    int step = 0; //攻撃開始から何ステップ経っているか
7|
8|    //コンストラクタ
9|    Enemy(double x0, double y0, int r0, Color c0, int hp0, int ctime0, int state0, int cooling0) {
10|        super(x0, y0, r0, c0, hp0, ctime0, state0, cooling0);
11|    }
12|
13|    Enemy(double x0, double y0, int r0, Color c0, int hp0, int aPattern0) {
14|        super(x0, y0, r0, c0, hp0, 0, 1, 0);
15|        aPattern = aPattern0;
16|    }
17|
18|    //攻撃処理
19|    public void action(Player player, LinkedList<Bullet> bullets) {
20|        if(state==1){
21|            if(aPattern == 1){
22|                AttackPattern.nerai(this, player, bullets, step, 30, 3.0);
23|            }
24|            else if(aPattern == 2){
25|                AttackPattern.uzumaki(this, player, bullets, step, 2, 0.1, 4.0);
26|            }
27|            else if(aPattern == 3){
28|                AttackPattern.housha(this, player, bullets, step, 10, 10, 3.0);
29|            }
30|        }
31|        step++;
32|    }
33|}
34|
```

自分で攻撃パターンを考えて、書いてみると面白いでしょう。

#### 4. 画像を表示する

自機の画像をまともな画像にしてみましょう。

##### ①画像の用意

ペイントなどのソフトで画像を作って、保存します（保存先は、C:¥Documents and Settings¥(ユーザー名)¥java¥ShootingGame¥bin¥）。画像は小さく（10\*10 ピクセルくらい）しましょう。

##### ②画像を読み込む

Shooting クラスの w や h を宣言しているところで、

```
Image img;
```

と書きましょう。

次に、init メソッドの中で、

```
img = getImage(getCodeBase(), “./jiki.jpg”);
```

と書きます。これで img に画像が読み込まれます。

##### ③画像の読み込みテスト

paint メソッドの中で、

```
g.drawImage(img, 100, 100, this);
```

と書いてください。実行したときに画像が表示されれば成功です。

##### ④Item クラスの編集

Item クラスの中に、paint メソッドをもう一つ追加します。

```
void paint(Graphics g, Image img, ImageObserver io){  
    if(state == 1){  
        g.drawImage(img, (int)(x-5), (int)(y-5), io);  
    }  
}
```

##### ⑤ Shooting クラスの編集

Shooting クラスの paint メソッドの、自機を描画している文

```
player.paint(g);
```

を、

```
player.paint(g, img, this);
```

と書き換えてください。

これで、自機が用意した画像に置き換わるはずです。

## 5. 敵の情報をファイルから読み込む

たくさんの種類の敵を出現させたり、敵の出現するタイミングや位置を指定したりすることで、ゲームを面白くしましょう。

これらの情報は、プログラムの中に書くと、プログラムが長くなってわけが分からなくなりがちで、別のファイルに分けて書くのが良いです。ここでは、別のファイルをどうやって扱うかを勉強します。

### ①読み込むファイルの形式

enemies.txtを作る。

このファイルには、敵の情報を書きます。

中身は、一行につき

[敵の名前] [画像名] [半径] [HP] [攻撃パターン番号]

をTabで区切って書いておきます。

### ②ファイルから読み込んだ敵のデータをあらわすクラスを作る

Enemydataクラスを作ります。

```
import java.awt.*;
public class Enemydata {
    String name;    //敵の名前
    Image img;      //画像
    int r;          //半径
    int hp;         //HP
    int aPattern;   //攻撃パターン番号
}
```

### ③読み込先となるリストを作る

Shooting.java内で

```
ArrayList<Enemydata> enemydata;
```

を作ります。ArrayListは今まで使ってきたLinkedListと似たようなものですが、LinkedListはデータの削除に強かったのに対し、こちらは、

「何番目のデータがほしい」という処理に適しています。

enemydata.add(なんか);    なんかを追加する

enemydata.get(i);        i番目のデータを取り出す

### ③読み込む

//敵データの読み込み

```
enemydata = new ArrayList<Enemydata>();
```

```
try {
```

```
    FileReader fr;
```

```
    fr = new FileReader("enemies.txt");
```

```
    StreamTokenizer st = new StreamTokenizer(fr);
```

```
    while(st.nextToken() != StreamTokenizer.TT_EOF) {
```

```
        Enemydata edata = new Enemydata();
```

```
        edata.name = st.sval;
```

```
        st.nextToken();
```

```
        edata.img = getImage(getCodeBase(), st.sval);
```

```
        st.nextToken();
```

```
        edata.r = (int)st.nval;
```

```
        st.nextToken();
```

```
        edata.hp = (int)st.nval;
```

```
        st.nextToken();
```

```
//ファイルを開く
```

```
//ファイルを読むカーソルみたいな
```

```
//ファイルの終わりまで繰り返す
```

```
//これに読んだデータを入れる
```

```
//文字列を読んで敵の名前とする
```

```
//次の単語へ
```

```
//文字列を読んでその名前の画像を開く
```

```
//次の単語へ
```

```
//数字を読んで敵の半径とする
```

```

        edata.aPattern = (int)st.nval;
        enemydata.add(edata); //読んだデータをリストに追加
    }
} catch (Exception e) {
}

```

#### ④Enemyクラスを改良する

Enemyクラス内のコンストラクタより前に、

Enemydata type;

さらに、コンストラクタのところに、

```

Enemy(double x0, double y0, Enemydata type0) {
    type = type0;
    x = x0;
    y = y0;
    r = type.r;
    hp = type.hp;
    state = 1;
    aPattern = type.aPattern;
}

```

という新しいコンストラクタを作ります。

#### ⑤使う

Shootingクラスの中のsetGameメソッドの中で、

```

Enemy e = new Enemy(w*Math.random(), h*Math.random()/3, enemydata.get(n));
enemies.add(e);

```

と書くことで、ファイルのn行目に書いた敵を出現させることができます。

#### 敵データの例

weak	teki.gif	8	2	1
midium	teki.gif	8	3	2
strong	teki.gif	8	4	3
vstrong	teki.gif	8	6	4



## 6. マップの情報をファイルから読み込む

### ①読み込むファイルの形式

map.txtを作る。このファイルには、どのタイミングで、どの敵を、どこに出現させるかを書きます。

中身は、一行につき

[出現させるステップ数] [敵の名前] [x] [y]

をTabで区切って書いておきます。

### ②ファイルから読み込んだ敵の出現タイミングの情報をあらわすクラスを作る

Mapdataクラスを作ります。

```
public class Mapdata {
    int step;           //出現させるステップ番号
    Enemydata edata;    //出現させる敵
    double x;           //x座標
    double y;           //y座標
}
```

### ③敵の名前（文字列）から敵のデータを検索する仕組みを作る

map.txtには、出現させる敵は文字列（String）で記述します。こうしたほうが後でステージを作るときに分かりやすいからです。しかし、その文字列とEnemydataを結び付けるには一工夫あります。

それには、HashMapというものを使います。これは、2つのデータを対応付けるもので、

たとえば英単語とその和訳、本のタイトルと本の中身、といった対応付けを行います。

ここでは、敵の名前と敵のデータを対応付けましょう。

まず、Shootingクラスのinitメソッドの手前に、

```
HashMap<String, Enemydata> hash;
```

```
ArrayList<Mapdata> mapdata;
```

と書きましょう。

次に、敵データの読み込みのところに、コメントがついている二行を追加します。

//敵データの読み込み

```
enemydata = new ArrayList<Enemydata>();
```

```
hash = new HashMap<String, Enemydata>(); //HashMapを用意
```

```
try {
```

```
    FileReader fr;
```

```
    fr = new FileReader("enemies.txt");
```

```
    StreamTokenizer st = new StreamTokenizer(fr);
```

```
    while(st.nextToken() != StreamTokenizer.TT_EOF) {
```

```
        Enemydata edata = new Enemydata();
```

```
        edata.name = st.sval;
```

```
        st.nextToken();
```

```
        edata.img = getImage(getCodeBase(), st.sval);
```

```
        st.nextToken();
```

```
        edata.r = (int)st.nval;
```

```
        st.nextToken();
```

```
        edata.hp = (int)st.nval;
```

```
        st.nextToken();
```

```
        edata.aPattern = (int)st.nval;
```

```
        enemydata.add(edata);
```

```
        hash.put(edata.name, edata);
```

//敵の名前と、このデータを対応付け

```
    }
```

```
} catch (Exception e) {
```

```
}
```

これで、敵のデータを読み込むところで、敵の名前とデータの対応付けをHashMapに格納します。

## ④読み込む

```
//マップデータの読み込み
mapdata = new ArrayList<Mapdata>();
try {
    FileReader fr;
    fr = new FileReader("map.txt");
    StreamTokenizer st = new StreamTokenizer(fr);
    while(st.nextToken() != StreamTokenizer.TT_EOF) {
        Mapdata mdata = new Mapdata();
        mdata.step = (int)st.nval;
        st.nextToken();
        String ename = st.sval;
        //HashMapから、ファイルから読んだ敵の名前に対応するEnemydataを探して持つ
        mdata.edata = hash.get(ename);
        st.nextToken();
        mdata.x = st.nval;
        st.nextToken();
        mdata.y = st.nval;
        mapdata.add(mdata);
    }
} catch (Exception e) {
}
```

## ⑤使う

今、mapdataリストには、ファイルに書いた敵の出現順にMapdataクラスのインスタンスが格納されています。

ファイルに書いたステップ番号に達したら、その敵をenemiesリストに追加する処理を書きましょう。

まず、Shootingクラスのinitメソッドより手前に、今mapdataの何番目のデータを見ているかを記録するpointer変数を宣言します。

```
int pointer;
```

次に、setGameメソッドでpointerを0に初期化します。

```
pointer = 0;
```

また、setGameメソッド内の、//敵はとりあえず10機と書いてあるfor文はもういらないので、コメントアウト（先頭に//）しましょう。

nextStepメソッドの中で、敵を出現させる命令を書きます。

```
while(pointer < mapdata.size() && mapdata.get(pointer).step <= step) {
    enemies.add(new Enemy(mapdata.get(pointer).x, mapdata.get(pointer).y, mapdata.get(pointer).edata));
    pointer++;
}
```

現在のステップ数が、次に出現させる敵の、出現予定ステップ数に達したら、enemiesリストに追加します。

最後に、paintメソッドを書き換えて、敵も○でなくて画像になるようにしましょう。

```
e.paint(g);
```

を消して、

```
e.paint(g, (e.type.img), this);
```

と書き換えてください。

マップデータの例

1	weak	50	50
1	weak	150	50
1	weak	350	50
1	weak	450	50
400	midium	100	10
400	midium	100	60
400	midium	400	10
400	midium	400	60
800	strong	10	10
800	strong	490	10
1600	vstrong	250	10
1600	vstrong	250	60

## 付録： 環境設定

今回の講座では Eclipse というソフトを使ってプログラミングをしました。  
プログラムを書くソフトを「エディタ」、プログラムを実行出来るようにするソフトを「コンパイラ」、  
バグをとる手助けをするソフトを「デバッガ」などといいますが、Eclipse はこれら全てが入った「統合開発環境」と呼ばれる優れたものです。

Java を使えるようにする流れは、

1. JDK をインストールする (Java の実行、開発環境)
  2. Eclipse をインストールする
  3. Eclipse を日本語化する
- です。

Eclipse のセットアップ方法などは、Google など検索すればたくさん出てくるので、ここでは割愛します。

例えば

「Eclipse で Java プログラミング超入門」

<http://www.atmarkit.co.jp/fjava/rensai3/eclipsejava01/eclipse01.html>

などをみれば丁寧に説明してあります。