

情報処理演習 1 1

鎌原淳三 kamahara@port.kobe-u.ac.jp

前回の課題

- ◎ `init()`, `move()`関数を編集して、個々のボールがランダムな方向に移動するようにせよ（ただし同じボールは途中で移動方向が変わってはいけない）
 - ◎ 基本課題：上下左右に移動する
 - ◎ 発展課題：ランダムに自由な角度で移動する
 - 移動する角度を変数で持つ(変数x,yなどと同じ)
 - `Math.sin()`, `Math.cos()`が使える(ラジアン)
 - π は、`Math.PI`
 - `Math.toRadian(角度)`でラジアンに変換できる
 - 移動量は整数 (`(int)`で整数化する)

<http://docs.oracle.com/javase/jp/6/api/java/lang/Math.html>

基本課題

```
public class Ball implements Runnable {  
    int x,y;  
    Color col;  
    int dir;  
  
    public void init() {  
        x = (int)(Math.random()*400);  
        y = (int)(Math.random()*400);  
        col = new Color(  
            (int)(Math.random()*256),  
            (int)(Math.random()*256),  
            (int)(Math.random()*256)  
        );  
        dir = (int)(Math.random()*4);  
        new Thread(this).start();  
    }  
}
```

```
public void move() {  
    if(dir==0) {  
        x = x + 5;  
    } else if (dir==1) {  
        x = x - 5;  
    } else if (dir==2) {  
        y = y + 5;  
    } else {  
        y = y - 5;  
    }  
}
```

init()で移動方向を決め、move()では次にどこに動くかを決める

追加課題

```
public class Ball implements Runnable {
    int x,y;
    Color col;
    double dir;

    public void init() {
        x = (int)(Math.random()*400);
        y = (int)(Math.random()*400);
        col = new Color(
            (int)(Math.random()*256),
            (int)(Math.random()*256),
            (int)(Math.random()*256)
        );
        dir = (int)(Math.random()*360);
        new Thread(this).start();
    }
}
```

角度を0～360度でランダム

*5は移動量。一定

```
public void move() {
    x = x + (int)(Math.cos(Math.toRadians(dir))*5);
    y = y + (int)(Math.sin(Math.toRadians(dir))*5);
}
```

別解

```
public class Ball implements Runnable {  
    int x,y;  
    Color col;  
    int dx, dy;  
  
    public void init() {  
        x = (int)(Math.random()*400);  
        y = (int)(Math.random()*400);  
        col = new Color(  
            (int)(Math.random()*256),  
            (int)(Math.random()*256),  
            (int)(Math.random()*256)  
        );  
        dx = (int)(Math.random()*10)-5;  
        dy = (int)(Math.random()*10)-5;  
        new Thread(this).start();  
    }  
}
```

-5から5未満の値をランダムに生成

```
public void move() {  
    x = x + dx;  
    y = y + dy;  
}
```

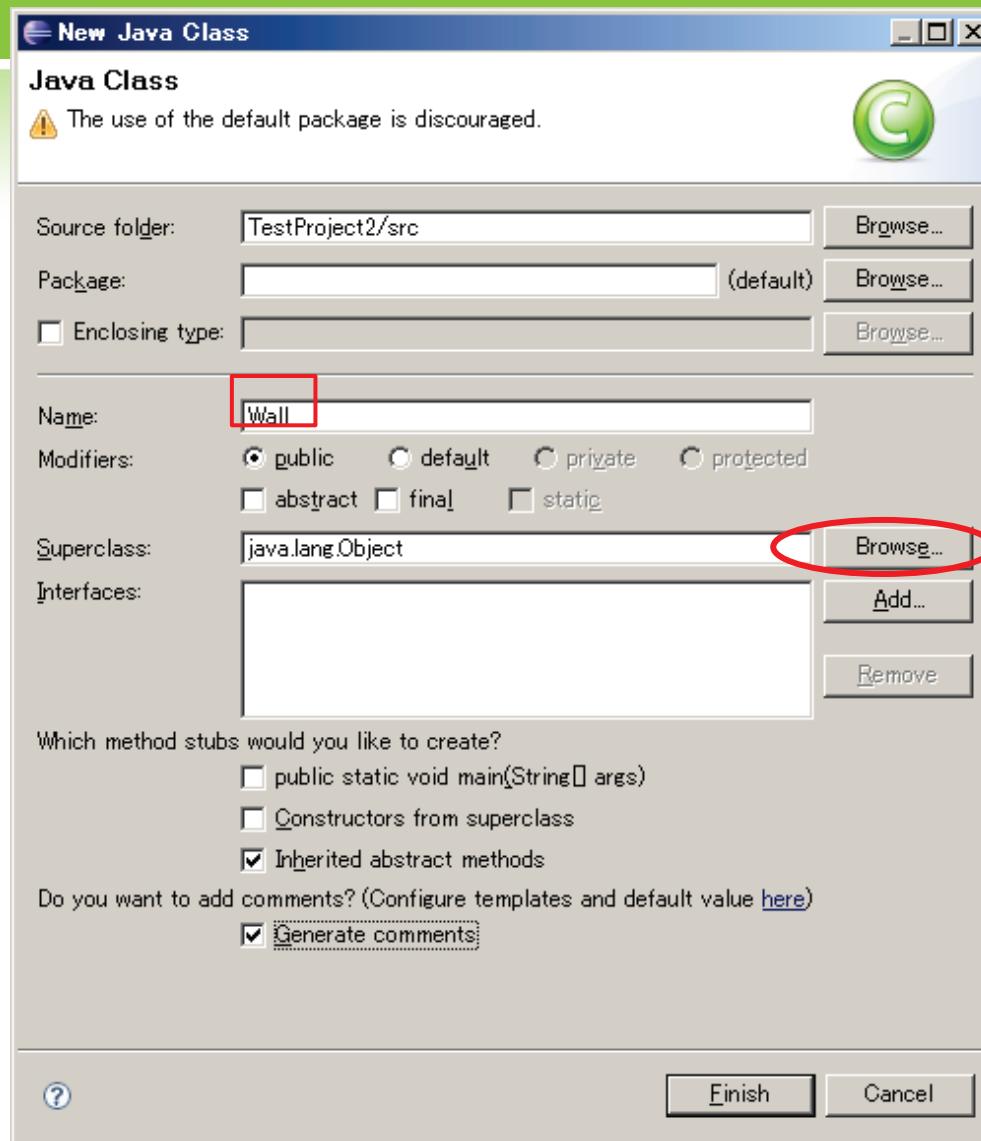
この場合、移動量(速度)はボールによって異なる

ボールが壁で反射するようをする

- ◎ 壁に当たった判定をどうするか
 - ◎ ボールの座標で判定する
 - ◎ 壁を作つてそれに当たつたかを判定する

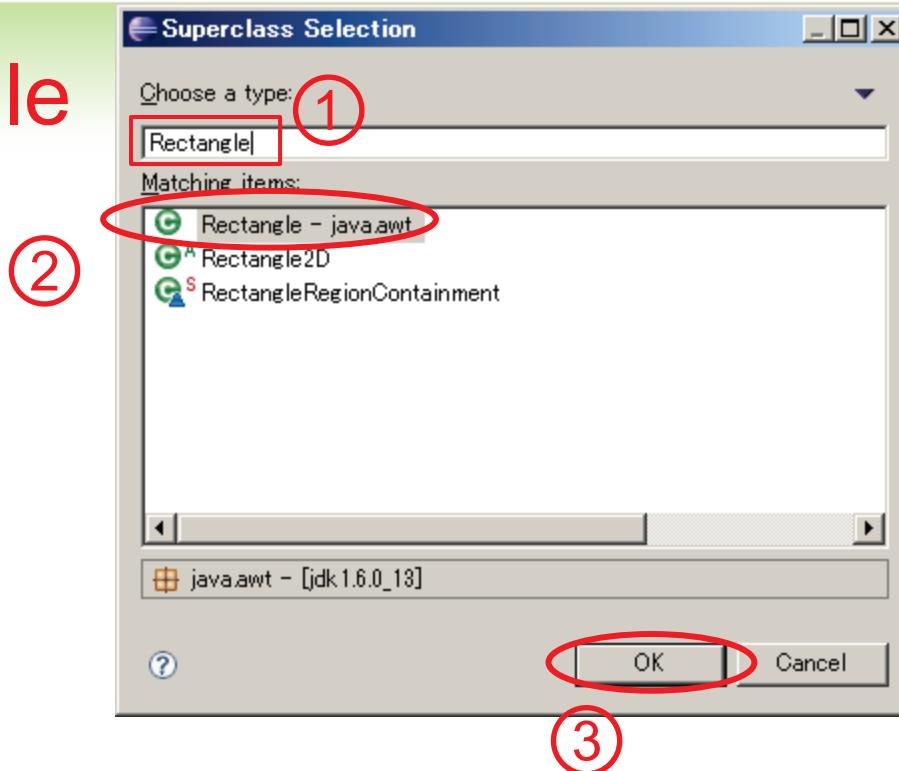
壁クラスを作る

Wall



スーパークラスに RECTANGLEを指定

Rectangle



下線部を確認して完了を押す



スケルトンが生成された

```
import java.awt.Rectangle;

+/* *
 *
 * @author kamahara
 *
 */
public class Wall extends Rectangle {

}
```

Rectangleクラスの子クラスとして生成

<http://docs.oracle.com/javase/jp/6/api/java.awt/Rectangle.html>

壁が縦か横を変数で持たせる

```
public class Wall extends Rectangle {  
    int dir; // 方向: 0=タテ 1=ヨコ  
}
```

コンストラクタを作る

- ◎ コンストラクタとは、インスタンスを生成する時に、行う処理を決めるところ
- ◎ super(...) で親クラスのコンストラクタを呼び出せる

```
public class Wall extends Rectangle {  
    int dir; // 方向: 0=タテ 1=ヨコ  
  
    public Wall(int x, int y, int width, int height, int direc) {  
        super(x, y, width, height);  
        dir = direc;  
    }  
}
```

クラス間の関係

Rectangle

```
int height;  
int width;  
int x;  
int y
```

Wall

```
int dir;
```



Wallクラスでは、dir変数だけでなく、Rectangleの中の
x, y, height, widthも扱うことができる

PAINT()メソッドを作る

```
public class Wall extends Rectangle {
    int dir; // 方向: 0=タテ 1=ヨコ

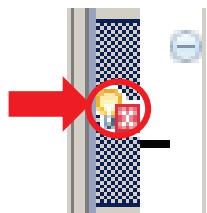
    public Wall(int x, int y, int width, int height, int direc) {
        super(x, y, width, height);
        dir = direc;
    }

    public void paint(Graphics g) {
    }
}
```



Graphicsをインポートする

壁の色を黒にする



Colorをインポートする

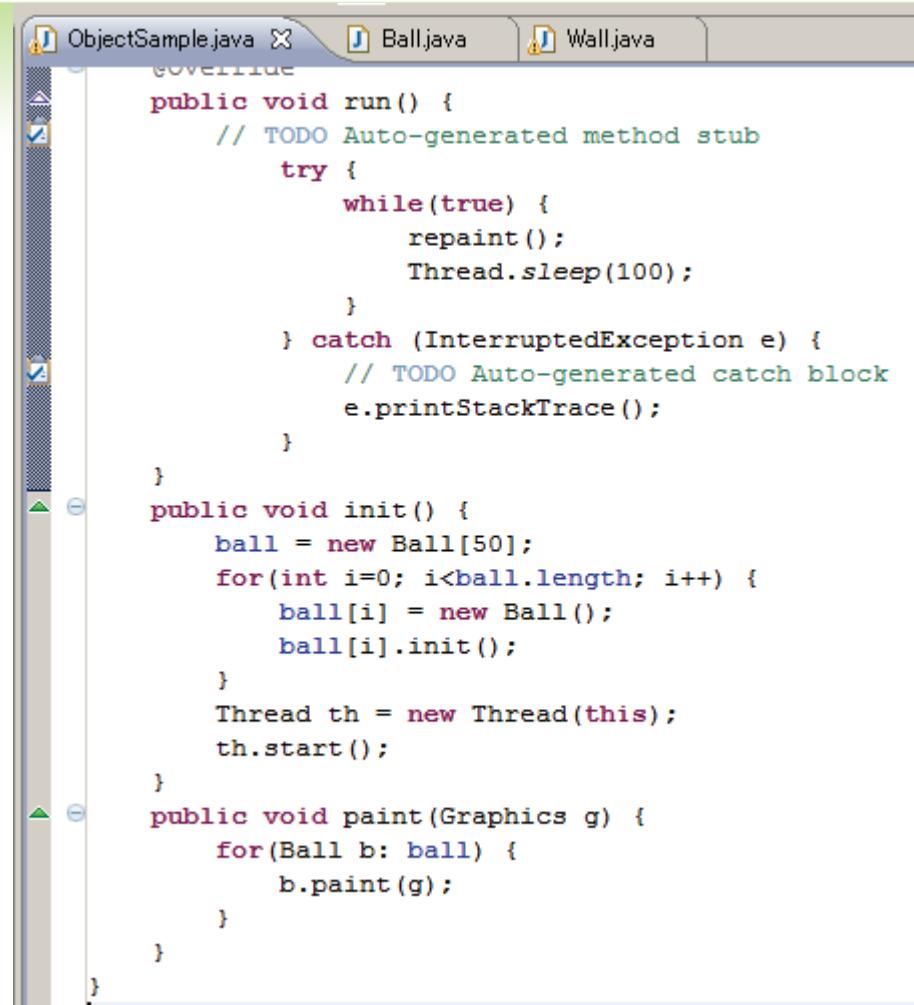
```
public void paint(Graphics g) {  
    g.setColor(new Color(0,0,0));  
}
```

色を変えたい場合は、この値を
変えればよい

FILLRECTで領域を塗りつぶす

```
public void paint(Graphics g) {  
    g.setColor(new Color(0,0,0));  
    g.fillRect(x, y, width, height);  
}
```

WALL.JAVAを保存して、 OBJECTSAMPLE.JAVAに切替



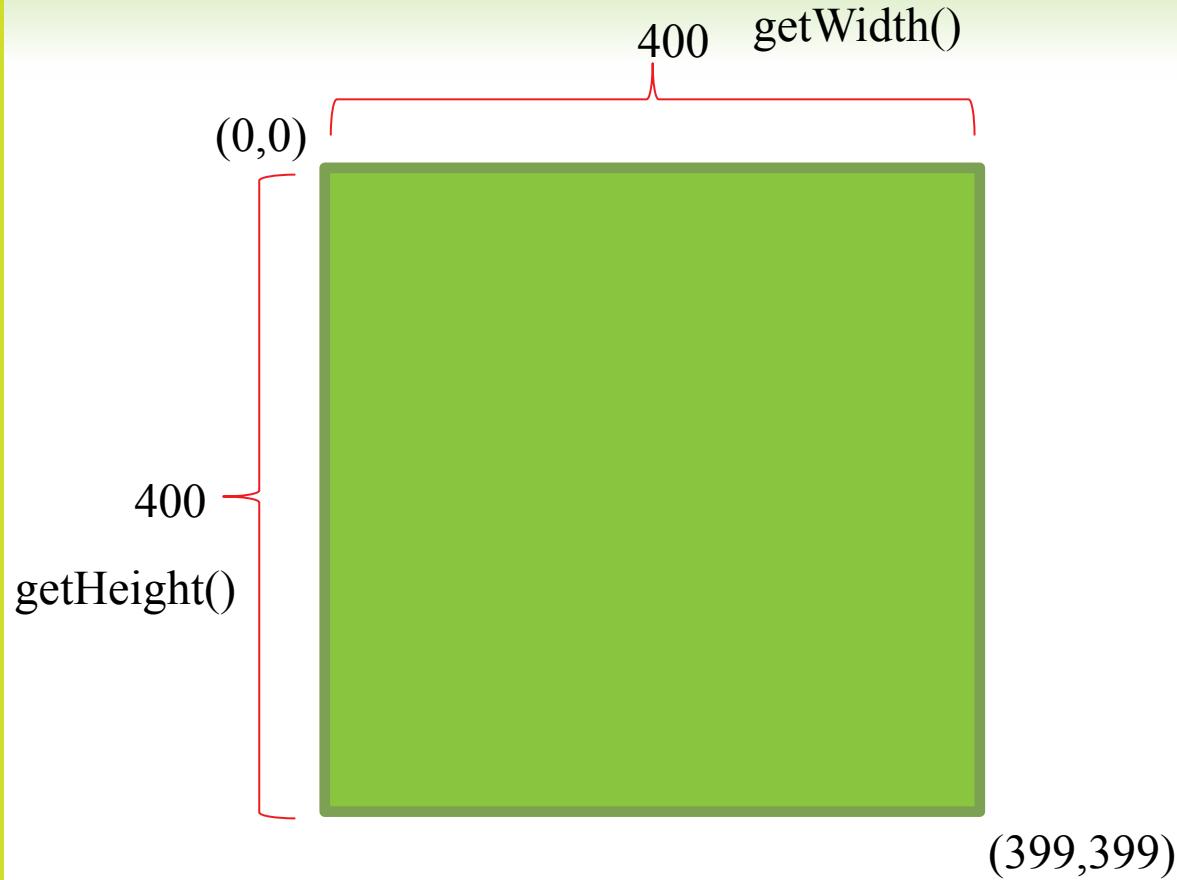
The screenshot shows a Java IDE interface with three tabs at the top: 'ObjectSample.java' (selected), 'Ball.java', and 'Wall.java'. The code editor displays the following Java code:

```
public class ObjectSample {
    public void run() {
        // TODO Auto-generated method stub
        try {
            while(true) {
                repaint();
                Thread.sleep(100);
            }
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public void init() {
        ball = new Ball[50];
        for(int i=0; i<ball.length; i++) {
            ball[i] = new Ball();
            ball[i].init();
        }
        Thread th = new Thread(this);
        th.start();
    }
    public void paint(Graphics g) {
        for(Ball b: ball) {
            b.paint(g);
        }
    }
}
```

WALLの配列を追加

```
public class ObjectSample extends Applet implements Runnable {  
    Ball[] ball;  
    Wall[] wall;
```

生成する壁の配置を考える



Appletの中ではgetWidth(),getHeight()で大きさが分かる

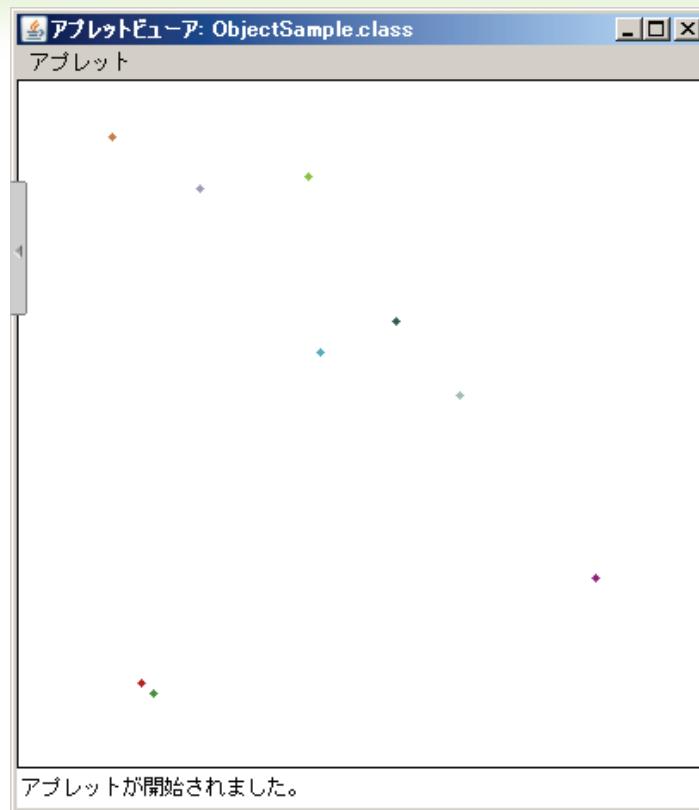
縦横の壁を生成する

```
public void init() {  
    wall = new Wall[4];  
    int w=getWidth(), hgetHeight();  
    wall[0] = new Wall( 0, 0, 1, h, 0);  
    wall[1] = new Wall( w-1, 0, 1, h, 0);  
    wall[2] = new Wall( 0, 0, w, 1, 壁の太さは1);  
    wall[3] = new Wall( 0, h-1, w, 1, 1);  
    ball = new Ball[10];  
    for(int i=0; i<ball.length; i++) {  
        ball[i] = new Ball();          方向を忘れないように  
        ball[i].init();  
    }  
    Thread th = new Thread(this);  
    th.start();  
}
```

壁を描画する

```
public void paint(Graphics g) {  
    for(Wall w: wall) {  
        w.paint(g);  
    }  
    for(Ball b: ball) {  
        b.paint(g);  
    }  
}
```

実行してみる



壁が描かれている。これだけではまだ跳ね返らない

BALLが壁をチェックできるようにする

```
public void init() {  
    wall = new Wall[4];  
    int w=getWidth(), h=getHeight();  
    wall[0] = new Wall( 0, 0, 1, h, 0);  
    wall[1] = new Wall( w-1, 0, 1, h, 0);  
    wall[2] = new Wall( 0, 0, w, 1, 1);  
    wall[3] = new Wall( 0, h-1, w, 1, 1);  
    ball = new Ball[10];  
    for(int i=0; i<ball.length; i++) {  
        ball[i] = new Ball(wall);  
        ball[i].init();  
    }  
    Thread th = new Thread(this);  
    th.start();  
}
```



コンストラクタを作成する方をダブルクリック

The screenshot shows an IDE interface with a code editor and a tool window.

Code Editor Content:

```
public void init() {  
    wall = new Wall[4];  
    int w=getWidth(), hgetHeight();  
    wall[0] = new Wall( 0, 0, 1, h, 0);  
    wall[1] = new Wall( w-1, 0, 1, h, 0);  
    wall[2] = new Wall( 0, 0, w, 1, 1);  
    wall[3] = new Wall( 0, h-1, w, 1, 1);  
    ball = new Ball[10];  
    for(int i=0; i<ball.length; i++) {  
        ball[i] = new Ball(wall); // Completion dropdown is shown here  
        ball[i].in  
    }  
    Thread th = ne  
    th.start();  
}  
public void paint()  
    for(Wall w: wa  
        w.paint(g)  
    }  
    for(Ball b: ba
```

Completion Dropdown:

- Remove argument to match 'Ball()'
- Create constructor 'Ball(Wall[])'

The second option, "Create constructor 'Ball(Wall[])'", is highlighted with a red oval.

Tool Window Content:

```
public Ball(Wall[] wall) {  
    // TODO Auto-generated constructor stub  
}  
  
public void init() {  
    x = (int)(Mathrandom()*400);  
    ...
```

Bottom Bar:

Problems Javadoc Declaration

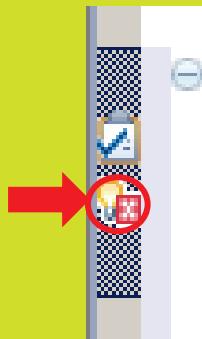
Press 'Tab' from proposal table or click for focus

BALL.JAVAにコンストラクタが作られる

```
public Ball(Wall[] wall) {  
    // TODO Auto-generated constructor stub  
}
```

フィールドを作成する

```
public Ball(Wall[] wall) {  
    // TODO Auto-generated constructor stub  
    this.wall = wall;  
}
```



ダブルクリック

A screenshot of an IDE interface showing Java code. The code defines a constructor for a class named 'Ball' that takes an array of 'Wall' objects as a parameter. The 'wall' parameter is highlighted with a red oval. A context menu is open at this position, with the first option, 'Create field 'wall' in type 'Ball'', also highlighted with a red oval. To the right, a preview pane shows the generated code:

```
Color col;
int dx, dy;
private Wall[] wall;
```

WALLがクラスの変数として追加された

```
public class Ball implements Runnable {  
    int x, y;  
    Color col;  
    int dx, dy;  
    private Wall[] wall;    追加された  
    public Ball(Wall[] wall) {  
        // TODO Auto-generated constructor stub  
        this.wall = wall;  
    }  
}
```

衝突をチェックするメソッドを作る

Ball.javaの中

```
    }
    public int checkCollision(int x, int y, int width, int height) {
    }
```

ボールの位置と大きさを矩形として指定するようにする
この段階ではエラーであってもよい(戻り値を指定しないため)

戻り値を作る

```
public int checkCollision(int x, int y, int width, int height) {  
    int ret = -1;  
    return ret;  
}
```

返す値は、壁に衝突していれば0か1
衝突していないければ-1を返すことにする

INTERSECS()で壁と交差しているか判定

```
public int checkCollision(int x, int y, int width, int height) {  
    int ret = -1;  
    for(Wall w: wall) {  
        if ( w.intersects(x, y, width, height) ) {  
            return w.dir;  
        }  
    }  
    return ret;  
}
```

衝突している場合は、壁の向きを返す

移動の際に衝突をチェックする

反射すると、移動方向が変化するようにする

壁が縦の場合

壁が横の場合

```
public void move() {  
    int col=checkCollision(x, y, 5, 5);  
    if (col==0) {  
        [ ]  
    } else if (col==1) {  
        [ ]  
    }  
}
```

適切な式を入力せよ

```
x = x + (int)(Math.cos(Math.toRadians(dir))*5);  
y = y + (int)(Math.sin(Math.toRadians(dir))*5);
```

}



別解の場合

```
x = x + dx;  
y = y + dy;
```

課題

- ◎ 真ん中に横の長さ 20 の壁を設けて、そこでも反射するようにせよ。

課題の提出

- ◎ Z:\workspace\TestProject2\src\Wall.java と SampleObject.java, Ball.java を添付ファイルとして提出すること。ファイル中にコメントして学籍番号が入っていること
- ◎ 件名：情報処理演習 1 1 学籍番号 名前
- ◎宛先：kamahara@port.kobe-u.ac.jp