

情報処理演習 1 2

鎌原 淳三 kamahara@port.kobe-u.ac.jp

自習

- ◎ 今日は出席はとりません。

前回：移動の際に衝突を チェックする

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

壁が縦の場合

壁が横の場合

適切な式を入力せよ

反射すると、移動方向が変化するようにする

反射するだけなら、向き を変えるだけ

```
public void move() {  
    int col=checkCollision(x, y, 5, 5);  
    if (col==0) {  
        dx = -dx;  
    } else if (col==1) {  
        dy = -dy;  
    }  
  
    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。

真ん中に長さ20の壁を追加

```
public void init() {  
    wall = new Wall[5];  
    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);  
    wall[4] = new Wall( w/2, h/2, 20, 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();  
}
```

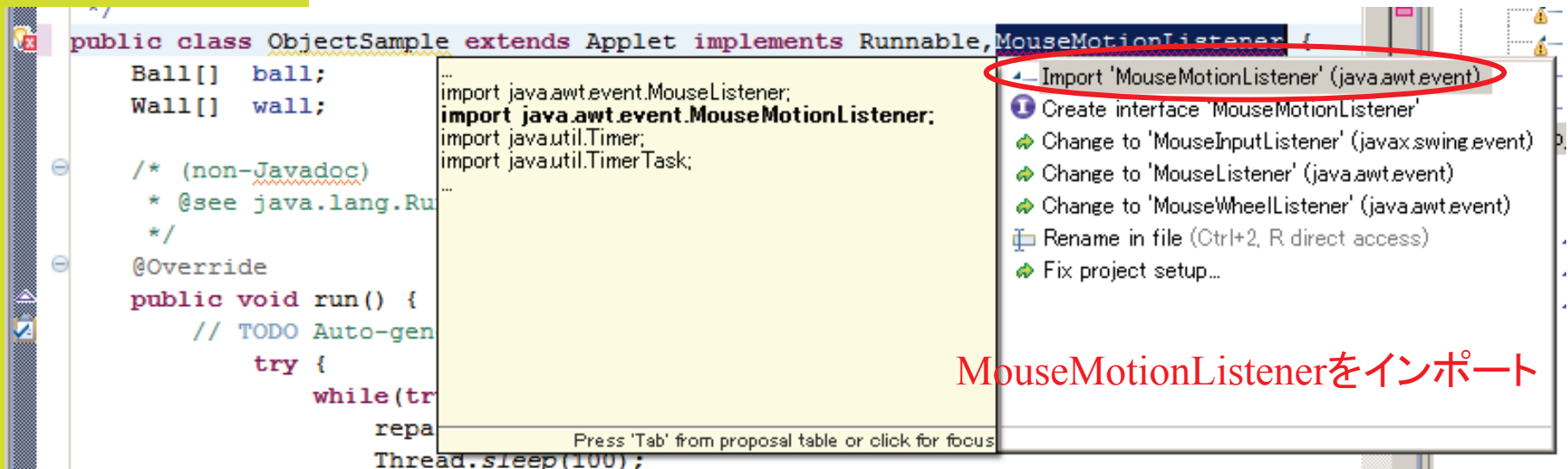
オブジェクト指向の特徴

- ◎ 壁の機能は同じなので、壁にぶつかったことによって反射するように作っていれば、壁を増やすのは非常に容易

マウスの動きで壁を動かしてみよう



```
/**
 * @author kamahara
 *
 */
public class ObjectSample extends Applet implements Runnable, MouseMotionListener {
    Ball[] ball;
    Wall[] wall;
```



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

    /* (non-Javadoc)
     * @see java.lang.Runnable:run()
     */
    @Override
    public void run() {
        // TODO Auto-generated method stub
        try {
            while (true) {
                repaint();
                Thread.sleep(100);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

import java.awt.event.MouseListener;
import java.awt.event.**MouseMotionListener**;
import java.util.Timer;
import java.util.TimerTask;

- Import 'MouseMotionListener' (java.awt.event)
- Create interface 'MouseMotionListener'
- Change to 'MouseListener' (java.awt.event)
- Change to 'MouseWheelListener' (java.awt.event)
- Rename in file (Ctrl+2, R direct access)
- Fix project setup...

Press 'Tab' from proposal table or click for focus

MouseMotionListenerをインポート

対応するメソッドを実装する

The diagram illustrates the process of implementing methods in a Java class. It starts with a code snippet for the `ObjectSample` class, which extends `Applet` and implements `Runnable` and `MouseMotionListener`. A red arrow points from a warning icon to the `MouseMotionListener` interface. A green arrow points down to a context menu and a list of methods to implement.

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

The context menu shows the following options:

- Add unimplemented methods
- + Add default serial version ID
- + Add generated serial version ID
- Make type 'ObjectSample' abstract
- 📁 Rename in file (Ctrl+2, R direct access)
- @ Add @SuppressWarnings 'serial' to 'ObjectSample'

The list of methods to implement is:

2 method(s) to implement:

- `java.awt.event.MouseMotionListener.mouseDragged()`
- `java.awt.event.MouseMotionListener.mouseMoved()`

実装されていないメソッドを実装する

下の方に追加された

```
}  
public void paint(Graphics g) {  
    for(Wall w: wall) {  
        w.paint(g);  
    }  
    for(Ball b: ball) {  
        b.paint(g);  
    }  
}  
  
@Override  
public void mouseDragged(MouseEvent e) {  
    // TODO Auto-generated method stub  
}  
  
@Override  
public void mouseMoved(MouseEvent e) {  
    // TODO Auto-generated method stub  
}  
}
```

マウスを押しながら動かした時の処理

マウスを動かした時の処理

真ん中の壁のx座標をマウスのX座標にする

```
-
@Override
public void mouseDragged(MouseEvent e) {
    // TODO Auto-generated method stub

}
@Override
public void mouseMoved(MouseEvent e) {
    // TODO Auto-generated method stub
    wall[4].x = e.getX();
}
}
```

ADDMOUSEMOTIONLISTENER で登録が必要

```
public void init() {  
    addMouseMotionListener(this);  
    wall = new Wall[5];  
    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);  
    wall[4] = new Wall( w/2, h/2, 20, 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();  
}
```

ここを修正すれば出現する
高さが変えられる

自動で動く壁を作ってみよう

New Java Class

Java Class

⚠ The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

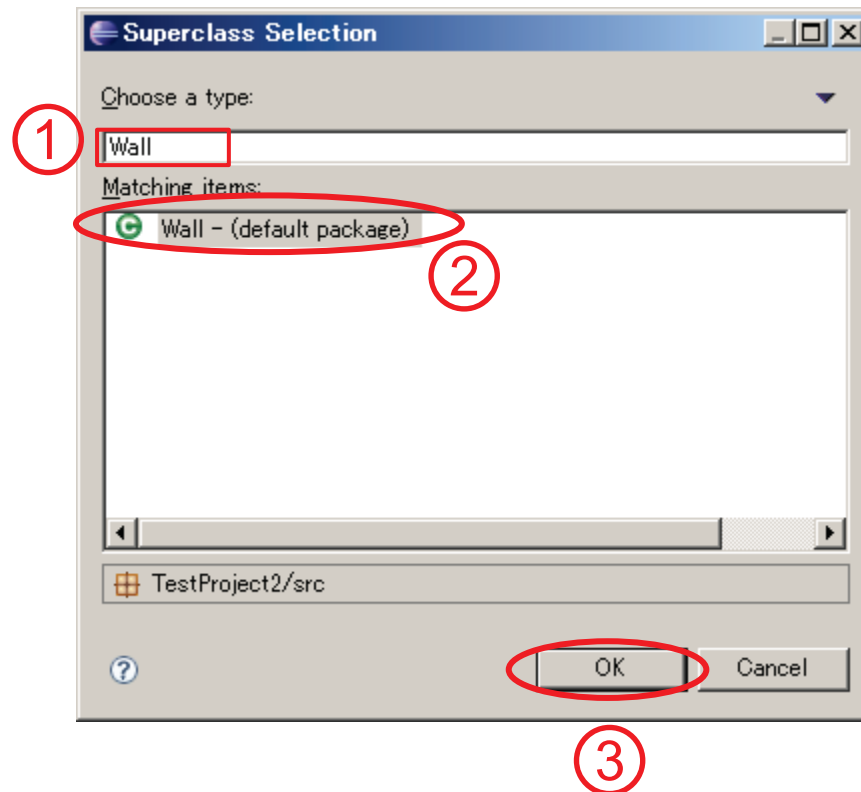
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

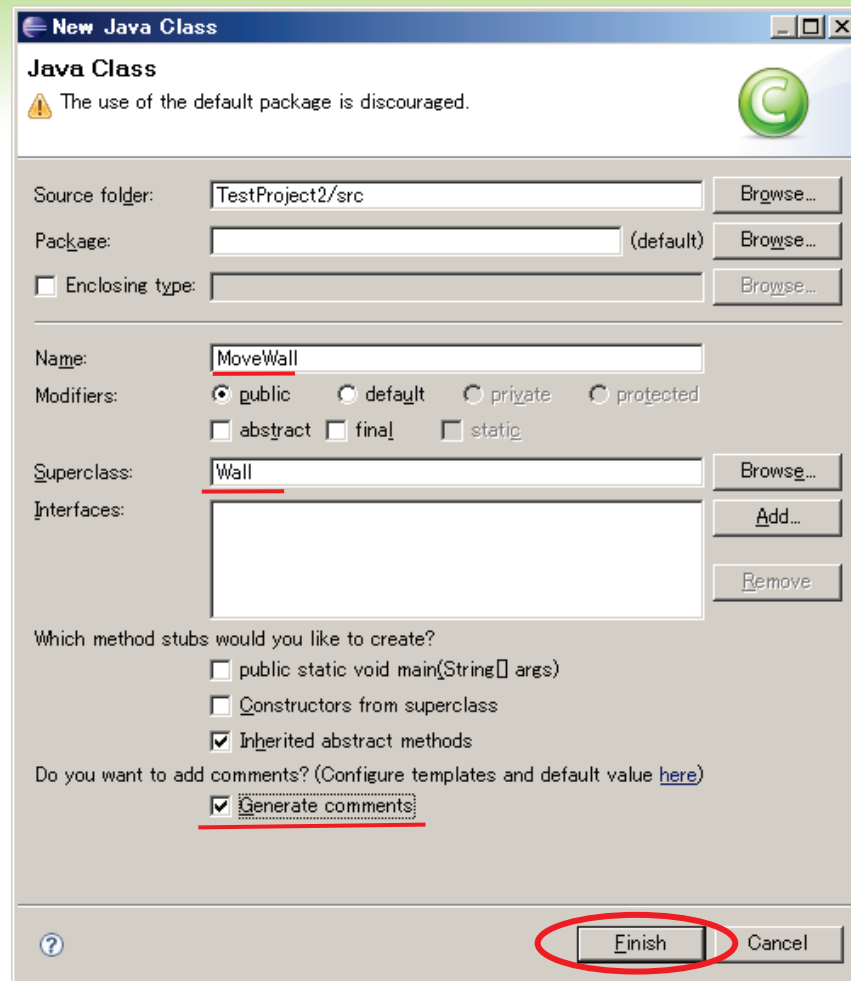
☐ Generate comments

新しいクラスを作る

親クラスはWALL



コメント生成にチェック



New Java Class

Java Class
⚠ The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

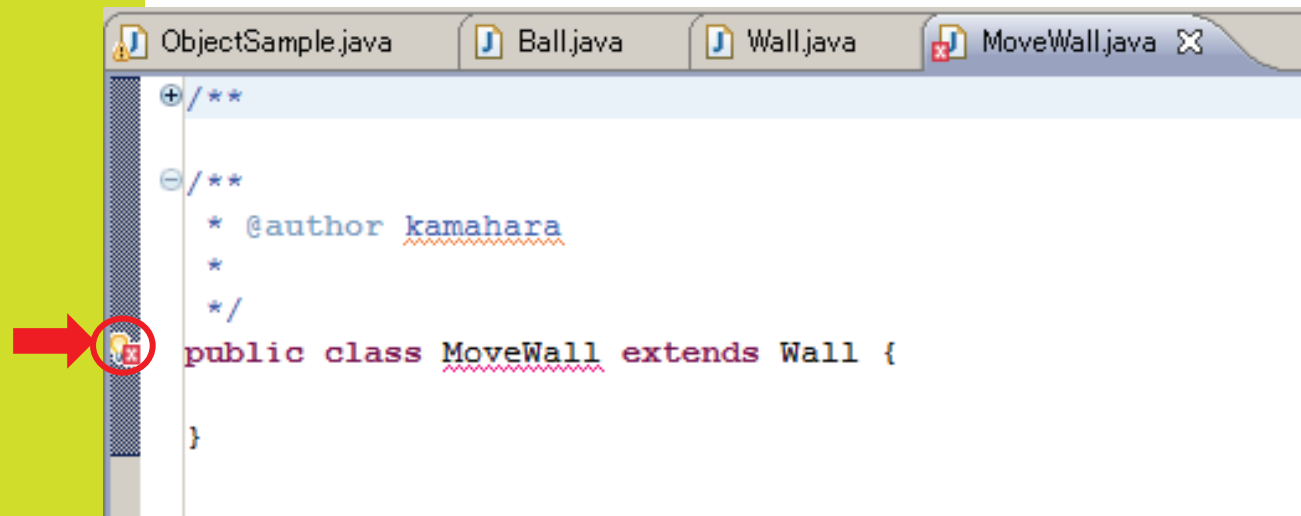
Which method stubs would you like to create?

- ☐ public static void main(String[] args)
- ☐ Constructors from superclass
- ☒ Inherited abstract methods

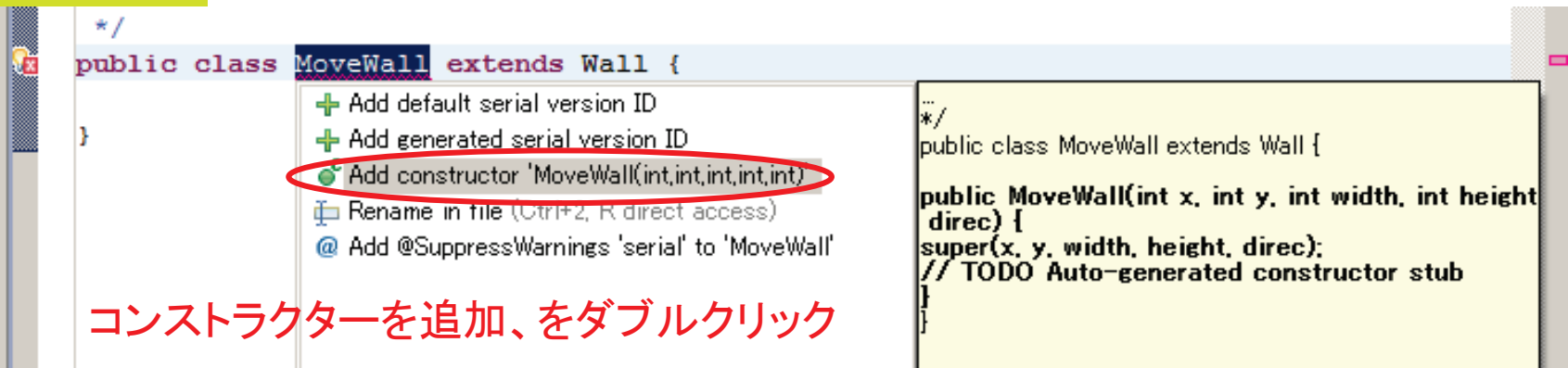
Do you want to add comments? (Configure templates and default value [here](#))

☒ Generate comments

MOVEWALLクラスの骨組み ができた



```
ObjectSample.java | Ball.java | Wall.java | MoveWall.java X
+ /**
- /**
  * @author kamahara
  *
  */
public class MoveWall extends Wall {
}
}
```



```
*/
public class MoveWall extends Wall {
}
}
```

- + Add default serial version ID
- + Add generated serial version ID
- Add constructor 'MoveWall(int,int,int,int,int)'
- 📄 Rename in file (Ctrl+Z, R direct access)
- @ Add @SuppressWarnings 'serial' to 'MoveWall'

```
*/
public class MoveWall extends Wall {
    public MoveWall(int x, int y, int width, int height,
        direc) {
        super(x, y, width, height, direc);
        // TODO Auto-generated constructor stub
    }
}
```

コンストラクターを追加、をダブルクリック

コンストラクターができた

```
public class MoveWall extends Wall {  
    public MoveWall(int x, int y, int width, int height, int direc) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
    }  
}
```



```
public class MoveWall extends Wall {  
    public MoveWall(int x, int y, int width, int height, int direc, int limit) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
    }  
}
```

移動できる横幅を追加する



LIMITをクラスの変数(フィールド)として作成する

②

```
public class MoveWall extends Wall {  
  
    public MoveWall(int x, int y, int width, int height, int direc, int limit) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
        this.limit = limit;  
    }  
}
```

①



③

```
public class MoveWall extends Wall {  
  
    public MoveWall(int x, int y, int width, int height, int direc, int limit) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
        this.limit = limit;  
    }  
}
```

フィールドを作成

- Create field 'limit' in type 'MoveWall'
- Change to 'dir'
- Change to 'height'
- Change to 'width'
- Change to 'x'
- Change to 'y'
- 📄 Rename in file (Ctrl+2, R direct access)

```
...  
private int limit;  
  
public MoveWall(int x, int y, int width, int height, int  
    direc, int limit) {  
    super(x, y, width, height, direc);  
    ...  
}
```

フィールドが作成された

```
/**
 * @author kamahara
 *
 */
public class MoveWall extends Wall {

    private int limit;

    public MoveWall(int x, int y, int width, int height, int direc, int limit) {
        super(x, y, width, height, direc);
        // TODO Auto-generated constructor stub
        this.limit = limit;
    }

}
```

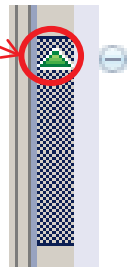
PAINTメソッドを書く

```
public class MoveWall extends Wall {  
  
    private int limit;  
  
    public MoveWall(int x, int y, int width, int height, int direc, int limit) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
        this.limit = limit;  
    }  
  
    public void paint(Graphics g) {  
        |  
    }  
}
```

Graphicsはインポートする

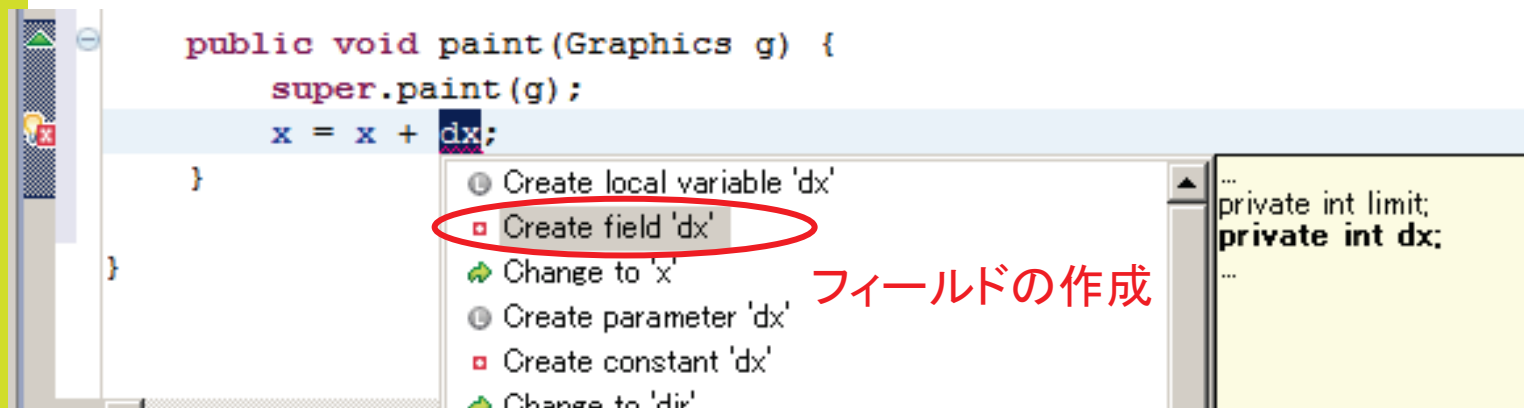
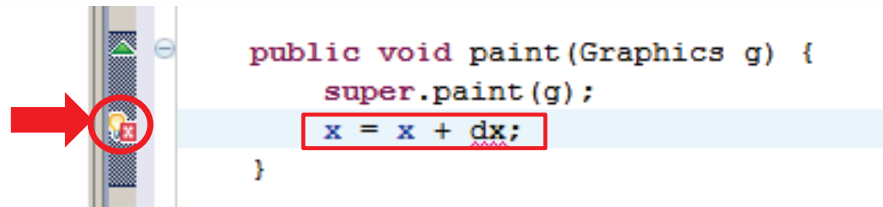
親クラスのPAINTメソッド を呼ぶ（壁を書く）

親のクラスを上書きする（オーバーライト）
ものは緑の△が出る（出ないと綴りが誤り）



```
public void paint(Graphics g) {  
    super.paint(g);  
}
```

ペイントの度に次の位置 を変更する



Dxもクラスの変数として 作成された

```
public class MoveWall extends Wall {  
  
    private int limit;  
    private int dx;  
  
    public MoveWall(int x, int y, int width, int height, int direc, int limit) {  
        super(x, y, width, height, direc);  
        // TODO Auto-generated constructor stub  
        this.limit = limit;  
    }  
  
    public void paint(Graphics g) {  
        super.paint(g);  
        x = x + dx;  
    }  
}
```

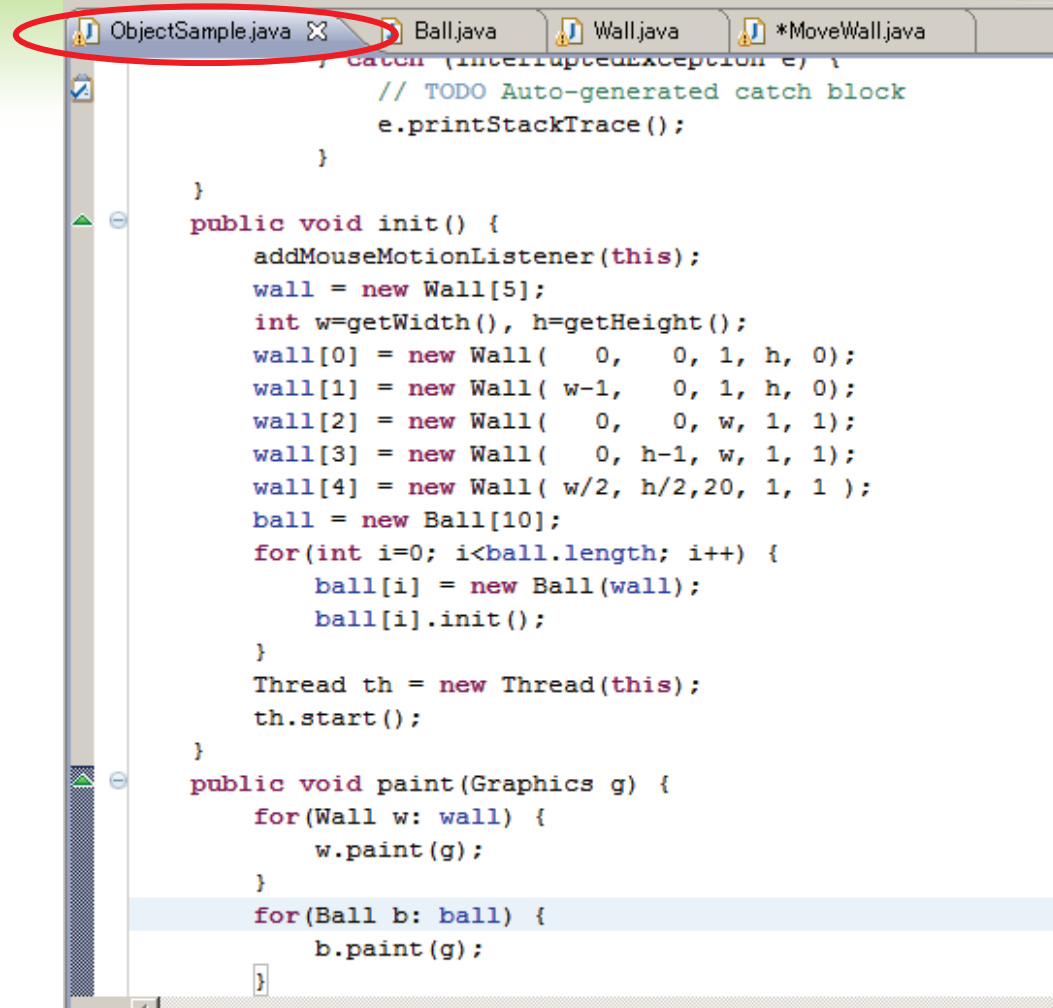
初期の移動量を設定しておく

```
public class MoveWall extends Wall {  
  
    private int limit;  
    private int dx = 2; ← 10くらいがよい  
  
    public MoveWall(int x, int y, int width, int height, int direction) {  
        super(x, y, width, height, direction);  
        // TODO Auto-generated constructor stub  
        this.limit = limit;  
    }  
  
    public void paint(Graphics g) {  
        super.paint(g);  
        x = x + dx;  
    }  
}
```

横をはみ出しそうになると移動する向きを変える

```
public void paint(Graphics g) {  
    super.paint(g);  
    x = x + dx;  
    if ( x<=0 || x+width >=limit ) {  
        dx = -dx;  
    }  
}
```

OBJECTSAMPLE.JAVAに切り替える



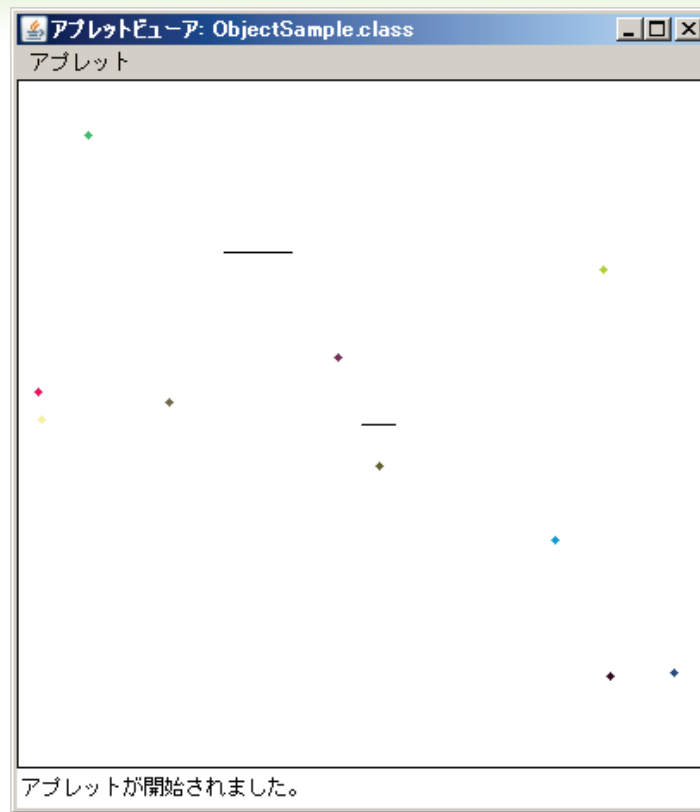
```
ObjectSample.java x Ball.java Wall.java *MoveWall.java
// catch (InterruptedException e) {
//     // TODO Auto-generated catch block
//     e.printStackTrace();
// }
}
public void init() {
    addMouseMotionListener(this);
    wall = new Wall[5];
    int w=getWidth(), h=getHeight();
    wall[0] = 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);
    wall[4] = new Wall( w/2, h/2, 20, 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();
}
public void paint(Graphics g) {
    for(Wall w: wall) {
        w.paint(g);
    }
    for(Ball b: ball) {
        b.paint(g);
    }
}
```

移動する壁(MOVEWALL)を 加える

```
public void init() {  
    addMouseMotionListener(this);  
    wall = new Wall[6];  
    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);  
    wall[4] = new Wall( w/2, h/2, 20, 1, 1 );  
    wall[5] = new MoveWall( 0, 100, 40, 1, 1, w);  
    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();  
}
```

移動する壁の高さは
適当に設定

実行してみる



得点を計算しよう

- ◎ 動く壁にボールが当たったら得点
 - ◎ ボールがどの壁に当たったか判定(A)
 - ◎ ボールが当たった回数を覚えておく(B)
 - ◎ すべてのボールが当たった回数を集計する(C)
- ◎ 得点を表示する必要がある
 - ◎ 得点を表示する(D)

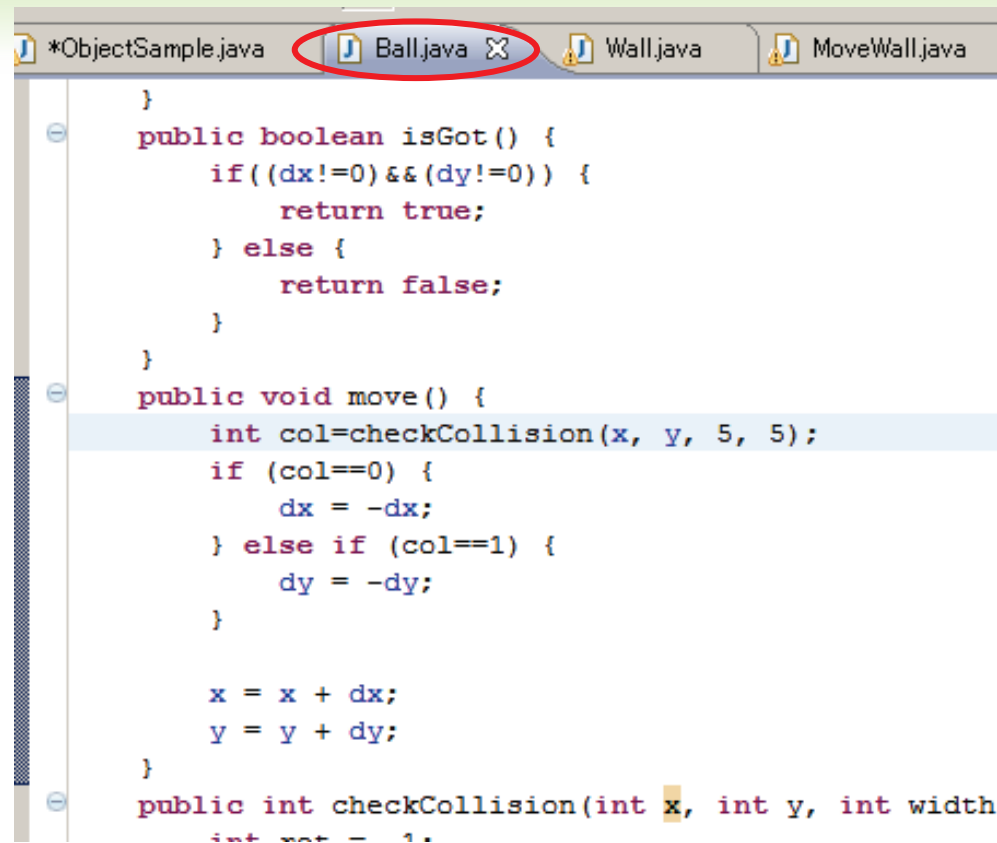
ボールがどの壁に当たったか判定

- ◎ どうするか
 - ◎ 横か縦で 1 か 0 かの種別がある
 - ◎ 一桁目を縦横の判定とし、二桁目を移動しているかどうかにする
 - ◎ 動く横の壁は 1 1 とする

動く壁の種別を1にする

```
public void init() {
    addMouseListener(this);
    wall = new Wall[6];
    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);
    wall[4] = new Wall( w/2, h/2, 20, 1, 1 );
    wall[5] = new MoveWall( 0, 100, 40, 1, 11, w);
    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();
}
```

BALL.JAVAに切替



```

}
public boolean isGot() {
    if ((dx!=0)&&(dy!=0)) {
        return true;
    } else {
        return false;
    }
}

public void move() {
    int col=checkCollision(x, y, 5, 5);
    if (col==0) {
        dx = -dx;
    } else if (col==1) {
        dy = -dy;
    }

    x = x + dx;
    y = y + dy;
}

public int checkCollision(int x, int y, int width
    int ret = 1;

```

今までのCOLはTYPEにし、 COLはTYPEから商を取る

```
,  
public void move() {  
    int type=checkCollision(x, y, 5, 5);  
    int col = type % 10;  
    if (col==0) {  
        dx = -dx;  
    } else if (col==1) {  
        dy = -dy;  
    }  
  
    x = x + dx;  
    y = y + dy;  
}
```

移動しているかどうかは MOVEに入れる

```
public void move() {  
    int type=checkCollision(x, y, 5, 5);  
    int col = type % 10;  
    int move = type / 10;  
    if (col==0) {  
        dx = -dx;  
    } else if (col==1) {  
        dy = -dy;  
    }  
  
    x = x + dx;  
    y = y + dy;  
}
```

当たっているのが移動する壁ならCOUNTを1増加

フィールド'count'を作成する



```
public void move() {  
    int type=checkCollision(x, y, 5, 5);  
    int col = type % 10;  
    int move = type / 10;  
    if (col==0) {  
        dx = -dx;  
    } else if (col==1) {  
        dy = -dy;  
    }  
    if(move==1) {  
        count++;  
    }  
  
    x = x + dx;  
    y = y + dy;  
}
```

(A)

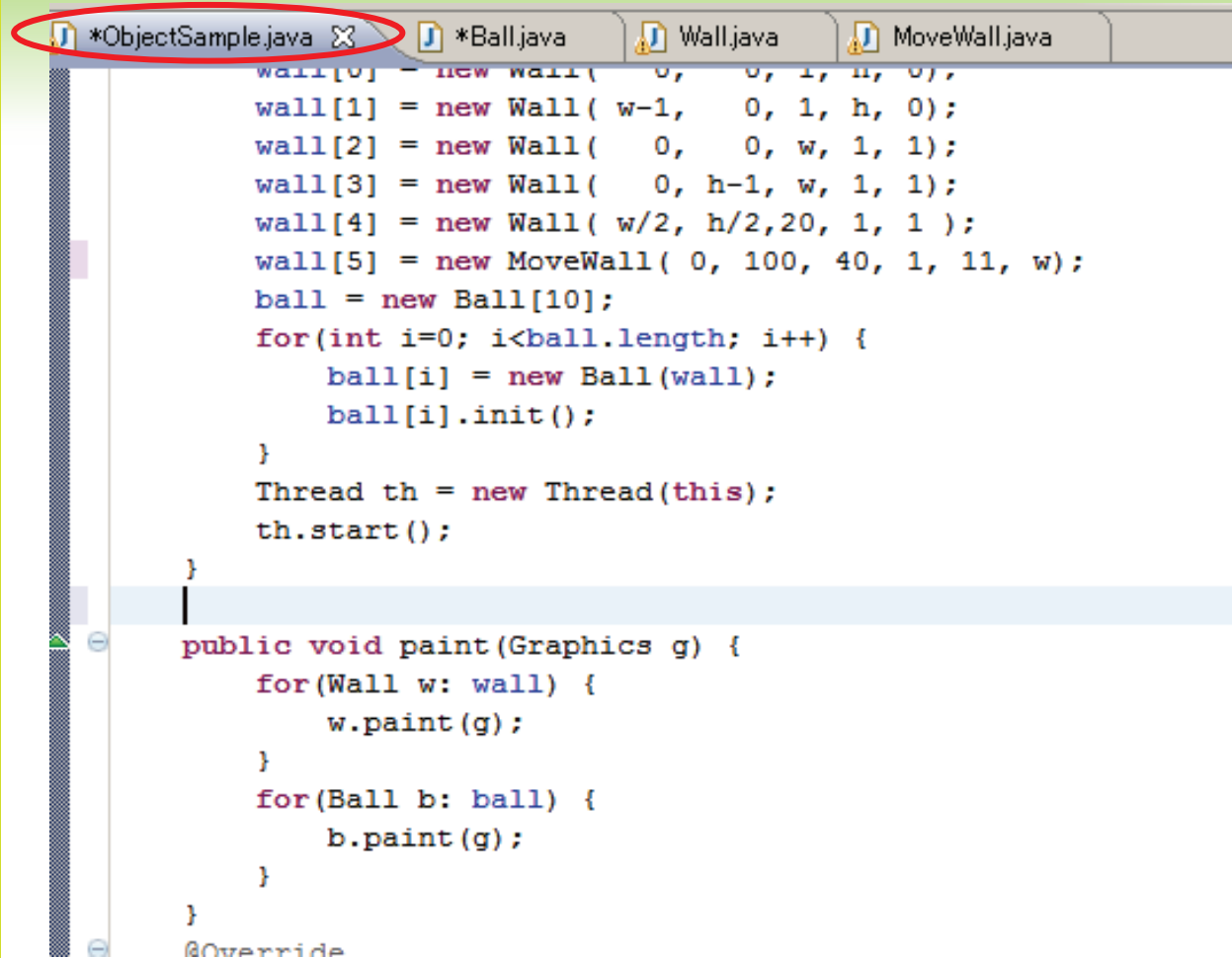
当たった回数が記録される変数できた

```
public class Ball implements Runnable {  
    int x,y;  
    Color col;  
    int dx, dy;  
    private Wall[] wall;  
    private int count;      (B)  
  
    public Ball(Wall[] wall) {
```

当たった回数を返すメソッドを作る

```
public class Ball implements Runnable {  
    int x,y;  
    Color col;  
    int dx, dy;  
    private Wall[] wall;  
    private int count;  
  
    public int getCount() {  
        return count;  
    }  
  
    public Ball(Wall[] wall) {
```

OBJECTSAMPLE.JAVAに切り替える



```
*ObjectSample.java × *Ball.java Wall.java MoveWall.java
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);
wall[4] = new Wall( w/2, h/2, 20, 1, 1 );
wall[5] = new MoveWall( 0, 100, 40, 1, 11, w);
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();
}

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

ボールのカウントを合計して集計する関数を作る

```
public void init() {
    addMouseListener(this);
    wall = new Wall[6];
    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);
    wall[4] = new Wall( w/2, h/2, 20, 1, 1 );
    wall[5] = new MoveWall( 0, 100, 40, 1, 11, w)
    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();
}

public int countBall() {
    int total=0;
    for(Ball b: ball) {
        total += b.getCount();
    }
    return total;
}

public void paint(Graphics g) {
    for(Wall w: wall) {
```

(C)

得点を表示する (DRAWSTRING)

```
public int countBall() {
    int total=0;
    for(Ball b: ball) {
        total += b.getCount();
    }
    return total;
}

public void paint(Graphics g) {
    for(Wall w: wall) {
        w.paint(g);
    }
    for(Ball b: ball) {
        b.paint(g);
    }
    g.drawString(String.valueOf(countBall()), 360, 360);
}

@Override
public void mouseDragged(MouseEvent e) {
```

(D)

自習課題

- ◎ ゲームとして以下のいずれかの機能を実装せよ
 - ◎ 上の壁に当たるとボールが消える（ペイントしないようにする）。消えたボールは当たっても得点はそれ以上増えない
 - ◎ 動く壁を増やし、その壁に当たると得点が2倍で計算される（できれば壁の色を変える）
 - ◎ 動く壁が方向を「時々」方向を変える
 - ◎ ブロック（壁）を表示して、ボールが当たるとブロックが消える（少し難しい）