Java講座

ブロック崩し

情報科学部コンピュータ科学科
2年　竹中　優

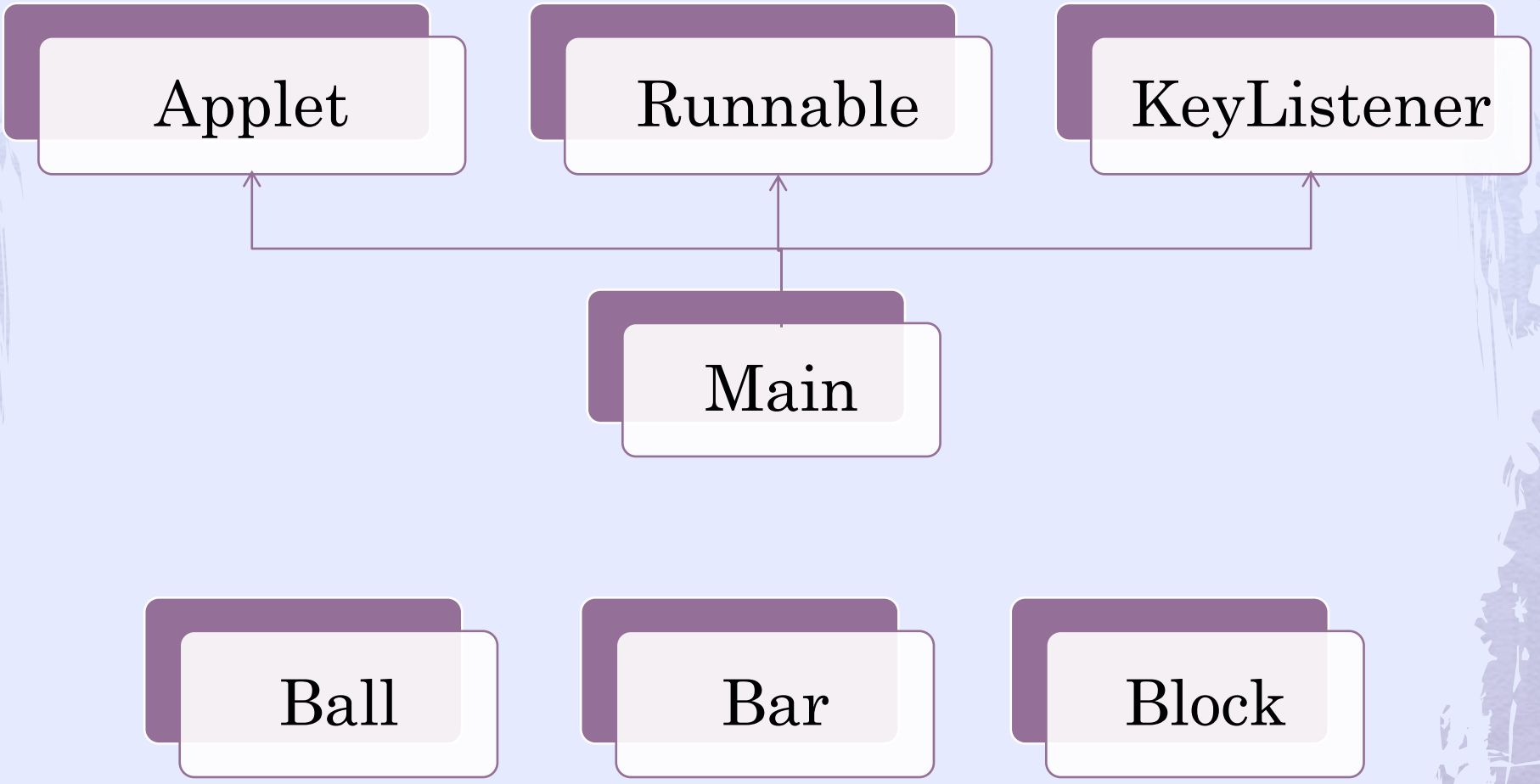
今回の内容

- ブロック崩しに必要なクラスを考えよう
  - クラス構造を考える
  - クラスを設計する
  - 実行してみる
- 当たり判定
- 実行クラスを完成させる

ブロック崩しに必要なクラスを考えよう

- とりあえず、ボールとバーとブロックを表すクラスが必要である。
- あとは、それらをまとめる実行クラス(Appletクラスのサブクラス)が必要である。
- クラス名は、それぞれBall, Bar, Block, Mainクラスで良いだろう。

クラス構造



クラス設計　Mainクラス

クラス設計　Mainクラス

- スーパークラス：Applet
  実装(implements)するインターフェイス：
  KeyListener, Runnable

- オーバーライドするAppletのメソッド
  public void init()
  public void paint(Graphics g)

- 実装しなければならないメソッド
  public void keyTyped(KeyEvent e)
  public void keyPressed(KeyEvent e)
  public void keyReleased(keyEvent e)
  public void run()

クラス設計　Mainクラス

- フィールドは、
  再描画ごとに動くボール　Ball ball
  左右矢印キーで動くバー　Bar bar
  画面に配置されるブロックの2次元配列
  　　Block[][] blocks
  一定ミリ秒間隔でrepaintメソッドを呼び、再描画するスレッド
  　　Thread repaintThread
  描画するシーンを表す変数
  　　int scene

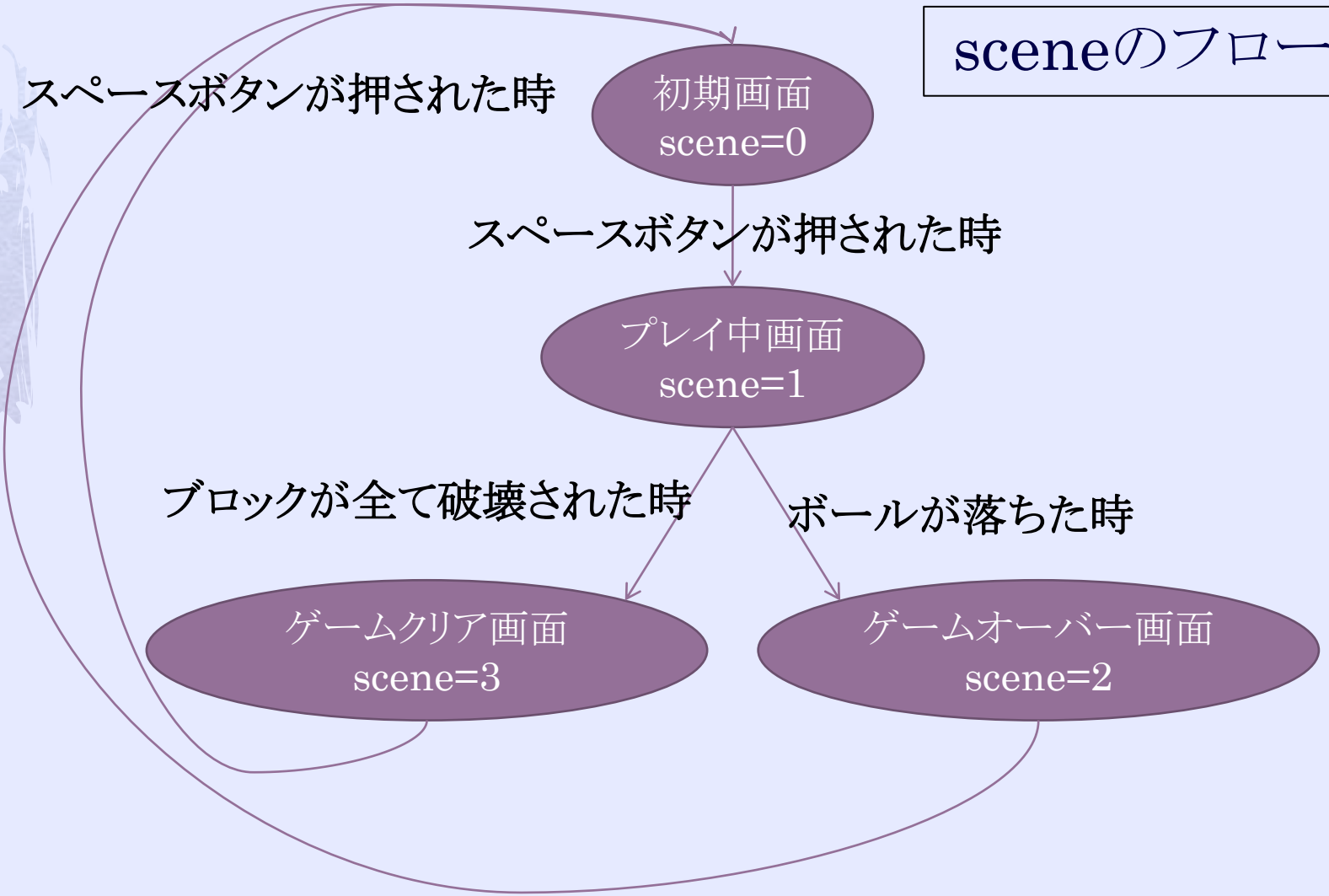
クラス設計　Mainクラス

- sceneについて
  0:初期画面
  1:プレイ中画面
  2:ゲームオーバー画面
  3:ゲームクリア画面

- paint(Graphics g)メソッドの中でsceneの値に対応する画面を描画するメソッドを呼び出す。

クラス設計　Mainクラス

sceneのフローチャート



クラス設計　Mainクラス

- メソッドは、
  Appletクラスのinit, paintメソッド
  KeyListenerインターフェイスの抽象メソッド
  　　keyTyped(KeyEvent e)
  　　keyPressed(KeyEvent e)
  　　keyReleased(KeyEvent e)
  Runnableインターフェイスの抽象メソッド
  　　run()
  scene=0の時にpaintメソッドが呼ぶメソッド
  　　ready(Graphics g)
  scene=1　　〃
  　　playing(Graphics g)
  scene=2　　〃
  　　gameover(Graphics g)
  scene=3　　〃
  　　gameclear(Graphics g)
- それぞれ記述が長いので、処理は後で記述する(名前だけ宣言しておこう)

クラス設計　Ballクラス

クラス設計　Ballクラス

- フィールドは、
  中心座標　double x, y
  再描画ごとの座標の変化量　double dx, dy
  半径　int r
  ゲームオーバーかどうか
  boolean isGameOver
  が必要となるだろう。
  そのほかは書いていくうちに随時追加していく。

クラス設計　Ballクラス

- メソッドは、
  ボールを座標x, yに描画するdrawメソッド
  ボールの座標を移動させるmoveメソッド
  ボールの当たり判定処理reflectメソッド
  (ぶつかっている場合は反射を行うメソッド)
  が必要となるだろう。
  そのほかは書いていくうちに随時追加していく。

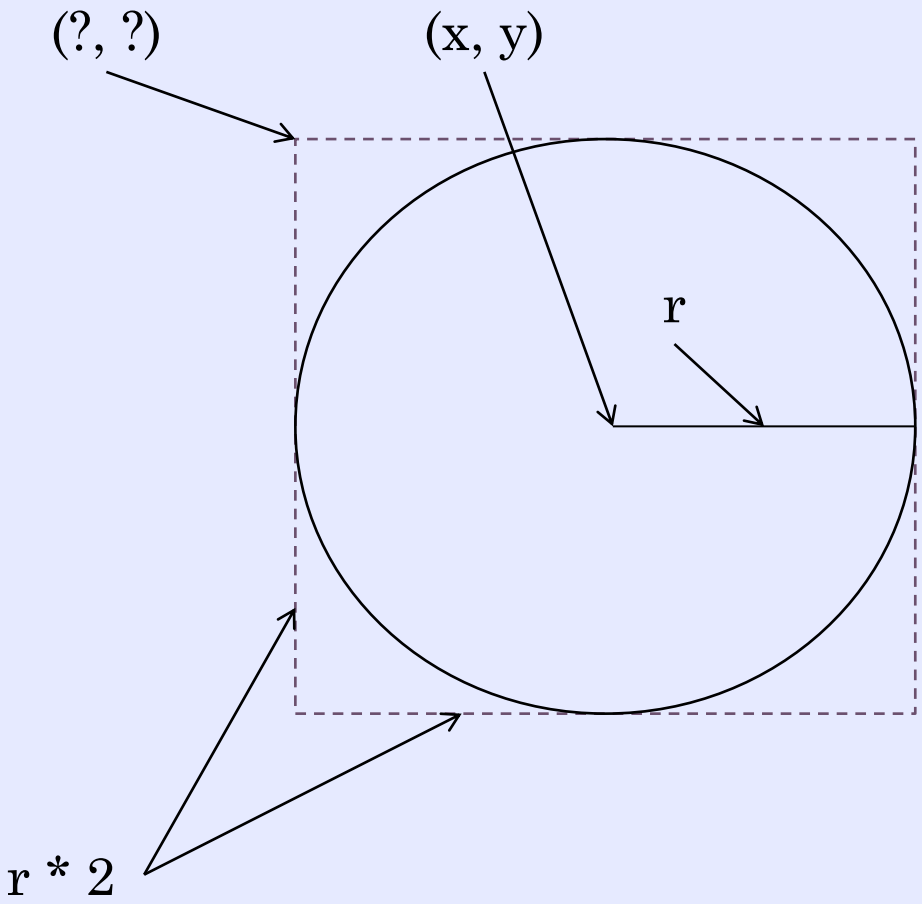
- 次項以降でdraw, move, reflectメソッドを定義していく。

クラス設計　Ballクラス

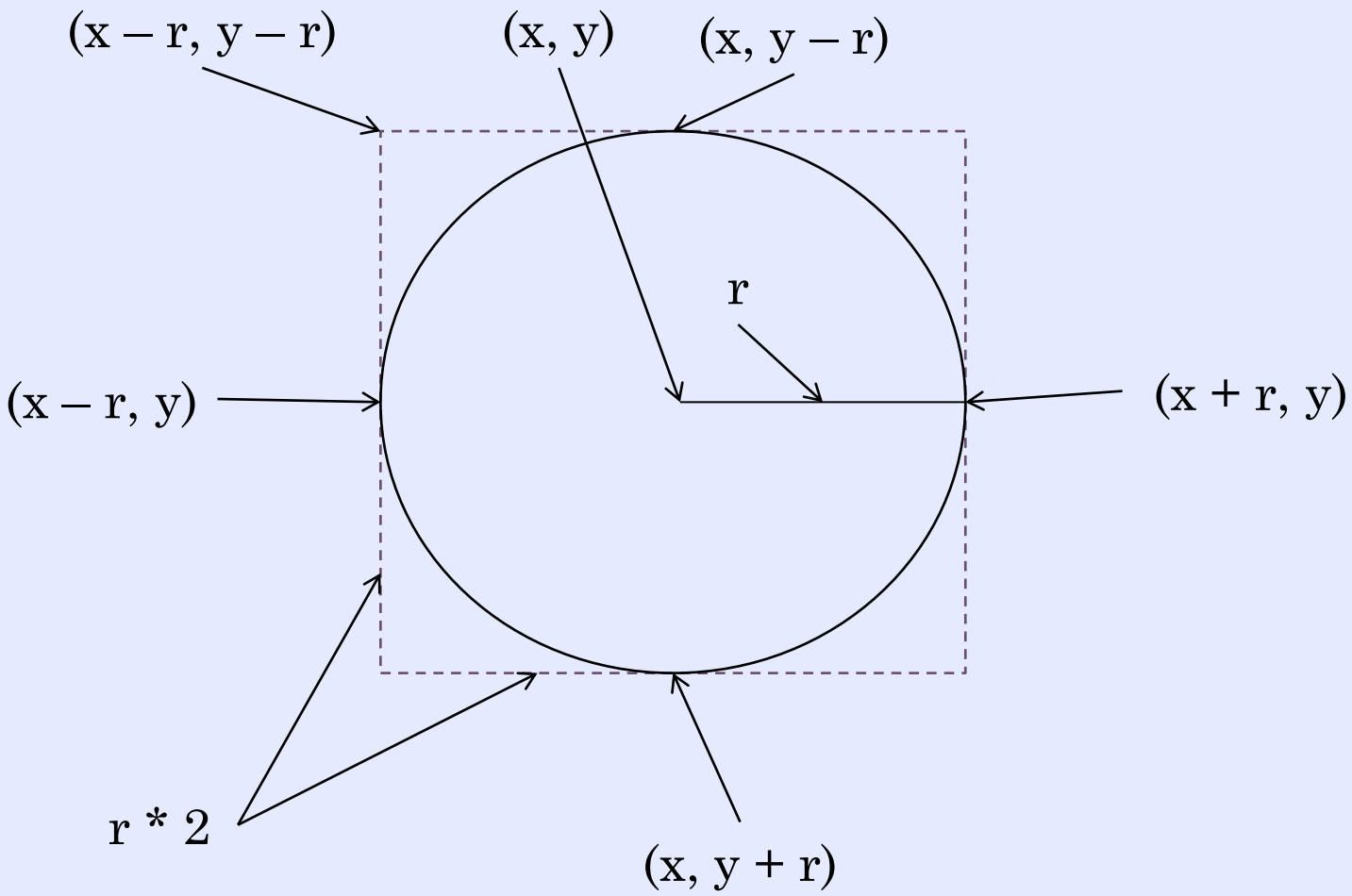
```
public void draw(Graphics g){
    g.drawOval((int)x – r,(int)y – r,r * 2,r * 2);
    //g.fillOval((int)x – r,(int)y – r,r * 2,r * 2);
}
```

- Graphicsオブジェクトを引数として受け取り、それを使って円を描画する。
- 今回はfillOvalではなく、drawOvalを使用する。
- 次項で座標指定の解説をする

クラス設計　Ballクラス



クラス設計　Ballクラス



クラス設計　Ballクラス

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

- x, yにそれぞれ再描画ごとの変化量dx, dyを加算する

クラス設計　Ballクラス

```
public void reflect(int screen_w, int screen_h,
                                    Bar bar, Block[][] blocks){
    /*壁との当たり判定を記述*/
    /*バーとの当たり判定を記述*/
    /*ブロックとの当たり判定を記述*/
}
```

- 壁、バー、ブロックとの当たり判定は、後に解説する

クラス設計　Barクラス

クラス設計　Barクラス

- フィールドは、
  左上の座標　double x, y
  幅　int width
  高さ　int height
  横移動する速さ(変化量)　double dx

クラス設計　Barクラス

- メソッドは、
  バーを描画するdrawメソッド
  バーを移動するmoveメソッド

- 次項以降でdraw, moveメソッドを定義していく。

クラス設計　Barクラス

```
public void draw(Graphics g){
    g.drawRect((int)x, (int)y, width, height);
}
```

- Ballクラスと同様にGraphicsオブジェクトを引数として受け取り、それを使って長方形を描画する。

クラス設計　Barクラス

```
public void move(boolean isRight){
        if(isRight)
                x += dx;
        else
                x -= dx;
}
```

- 引数isRightがtrueの場合、バーを右に移動させ、falseの場合、バーを左に移動させる。
- すなわち、isRightは矢印キーが「→」の場合、true「←」の場合、falseとする。

クラス設計　Blockクラス

クラス設計　Blockクラス

- 同じ長方形なので、Barクラスとほぼ同じ

- フィールドは、
  左上の座標　double x, y
  幅　int width
  高さ　int height
  既に壊されたかどうか　boolean isBroken

クラス設計　Blockクラス

- メソッドは、
  ブロックを描画するdrawメソッド

- 次項以降でdrawメソッドを定義していく。

クラス設計　Blockクラス

```
public void draw(Graphics g){
    if(isBroken){
        g.drawRect((int)x, (int)y, width, height);
    }
}
```

- Ball, Barクラスと同様にGraphicsオブジェクトを引数として受け取り、それを使って長方形を描画する。
- フィールドisBrokenがtrueの場合は描画し、falseの場合は描画しない。

Mainクラス　フィールドの宣言

```
12      /** ボール */
13      private Ball ball;
14
15      /** バー */
16      private Bar bar;
17
18      /** ブロック */
19      private Block[][] blocks = new Block[10][10];
20
21      /** 一定ミリ秒間隔ごとに画面を再描画するスレッド */
22      private Thread repaintThread;
23
24      /**
25       * シーンを表す変数<br>
26       * 0:初期画面<br>
27       * 1:プレイ中画面<br>
28       * 2:ゲームオーバー画面<br>
29       * 3:ゲームクリア画面<br>
30       */
31      private int scene = 1;
```

Mainクラス　initメソッド

```
22      @Override
23      public void init(){
24          //画面の初期サイズを指定
25          setSize(300, 400);
26          //画面のバックグラウンドカラーを黒色に指定
27          setBackground(Color.BLACK);
28
29          ball = new Ball(getWidth() / 2, getHeight() - 50, 0.5, -0.5);
30          bar = new Bar(getWidth() / 2 - 60 / 2, getHeight() - 30, 60, 5, 3);
31
32          int width = 28;
33          int height = 10;
34          for(int i = 0; i < blocks.length; i++){
35              for(int j = 0; j < blocks[i].length; j++){
36                  int space = 2;//余白
37                  int x = j * (width + space);
38                  int y = (i + 1) * (height + space);
39                  blocks[i][j] = new Block(x, y, width, height);
40              }
41          }
42
43          /*
44           * Runnableインターフェイスを実装したこのクラスを指定して、
45           * Threadオブジェクトを生成する
46           */
47          repaintThread = new Thread(this);
48          repaintThread.start();//再描画するスレッドをスタート
49      }
```

Mainクラス　paintメソッド

```
61      @Override
62      public void paint(Graphics g){
63          g.setColor(Color.WHITE);//描画色を白に変更
64          switch(scene){
65          case 0:
66              ready(g);//初期画面
67              break;
68          case 1:
69              playing(g);//プレイ中画面
70              break;
71          case 2:
72              gameover(g);//ゲームオーバー画面
73              break;
74          case 3:
75              gameclear(g);//ゲームクリア画面
76              break;
77          }
78      }
```

Mainクラス　playingメソッド

```
87      /**
88       * プレイ中の画面を描画するメソッド<br>
89       * scene = 1のときに呼ばれる
90       * @param g
91       */
92      private void playing(Graphics g){
93          ball.draw(g);
94          bar.draw(g);
95          for (int i = 0; i < blocks.length; i++) {
96              for (int j = 0; j < blocks[i].length; j++) {
97                  blocks[i][j].draw(g);
98              }
99          }
100     }
```

Mainクラス　runメソッド

```
118     @Override
119     public void run(){
120         while(true){
121             try{
122                 Thread.sleep(10);
123                 repaint();
124
125
126                 //次の描画に備えてボールを移動
127                 ball.move();
128                 //ボールの当たり判定処理
129                 ball.reflect(getWidth(), getHeight());
130             }
131             catch(InterruptedException e){
132
133             }
134         }
135     }
```

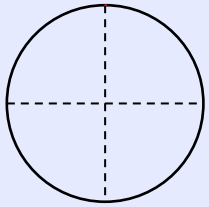
当たり判定

当たり判定

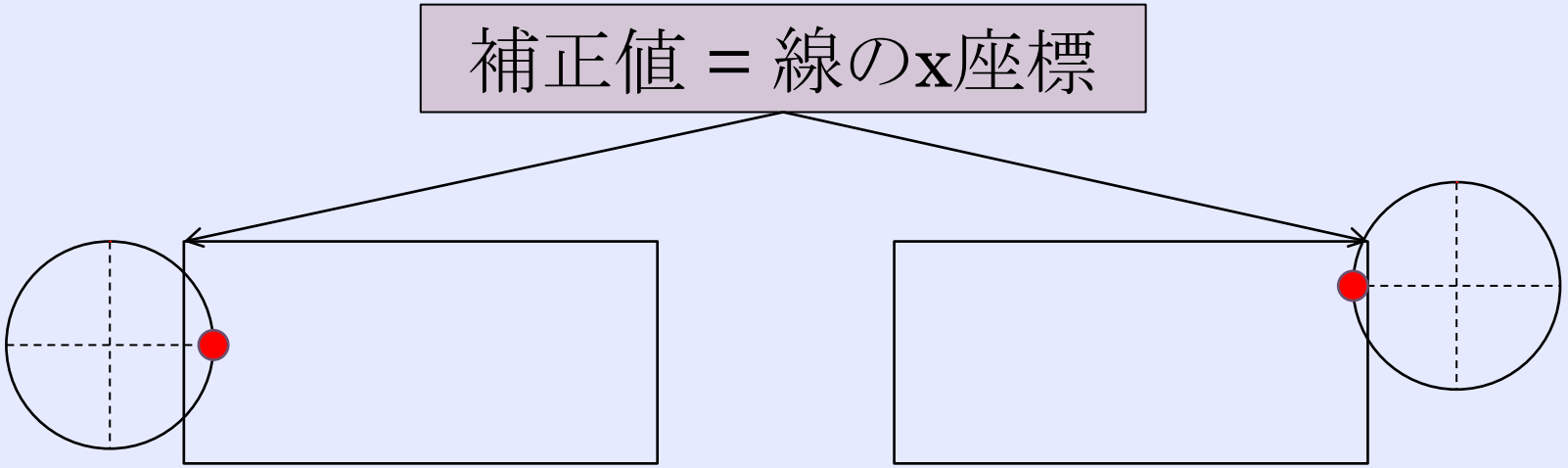
- Ballクラスのreflect()メソッドに記述する。
- 今回は、円と長方形の当たり判定のみ
  (壁との当たり判定は実装済み)
- 次項からはブロックとボールの当たり判定を考える。

当たり判定

- 円=ボール
- 長方形=ブロック



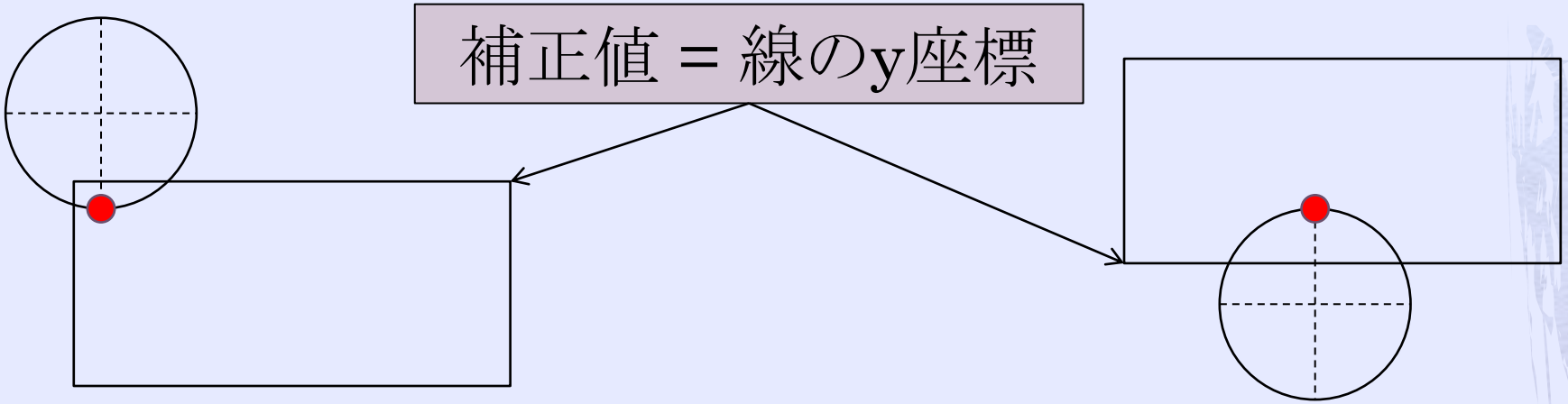
当たり判定



- 赤い点が長方形の左右どちらかに接する、または入り込んでいる場合、

```
ball.dx *= -1;
ball.x = 補正値;
```

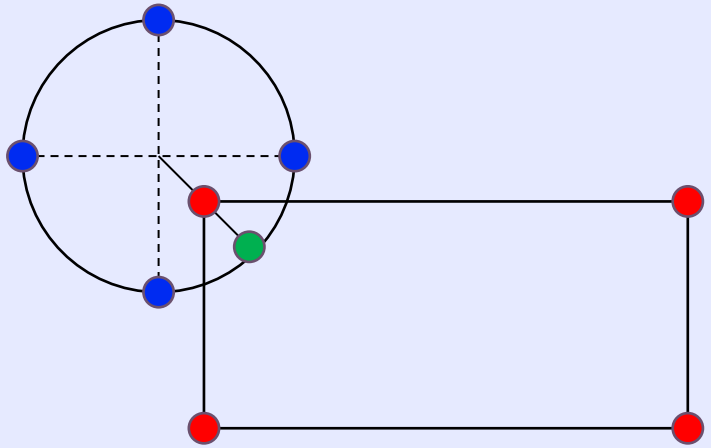
当たり判定



- 赤い点が長方形の上下どちらかに接する、または入り込んでいる場合、

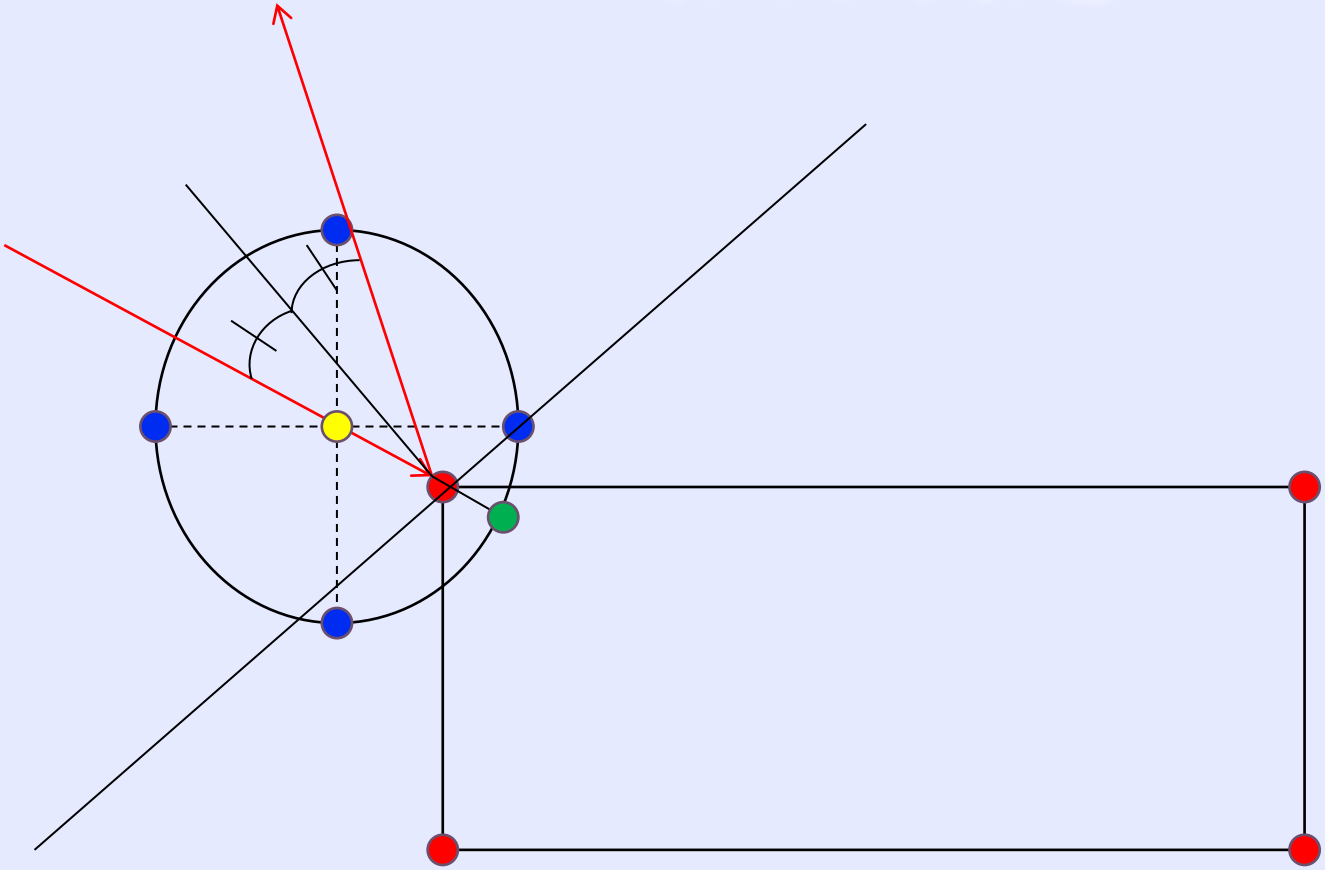
```
ball.dy *= -1;
ball.y = 補正値;
```

当たり判定



- 赤い点が円に接する、または入り込んでいる場合、
  ボールの座標を赤い点と緑の点に接するように補正し、反射する。

当たり判定



これは難しい。。。

当たり判定

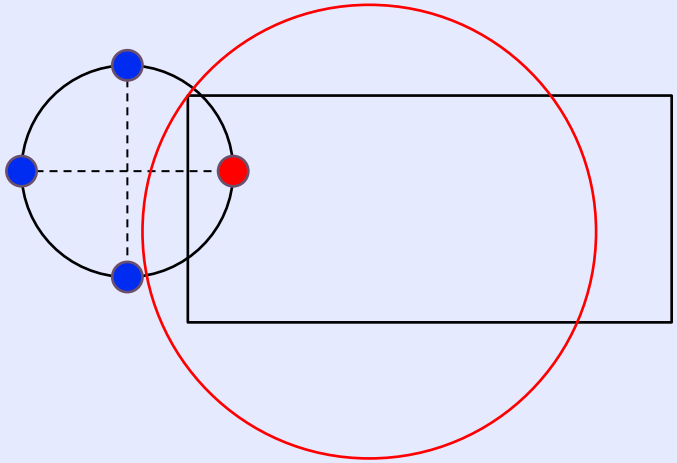
- 前項の反射を実装するのは難しいので、とりあえず、

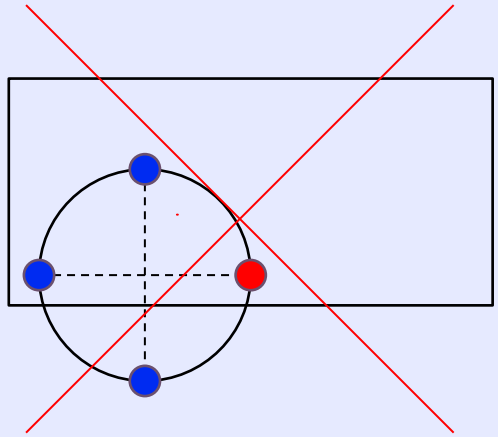
```
ball.dx *= -1;
ball.x = 補正値;
```

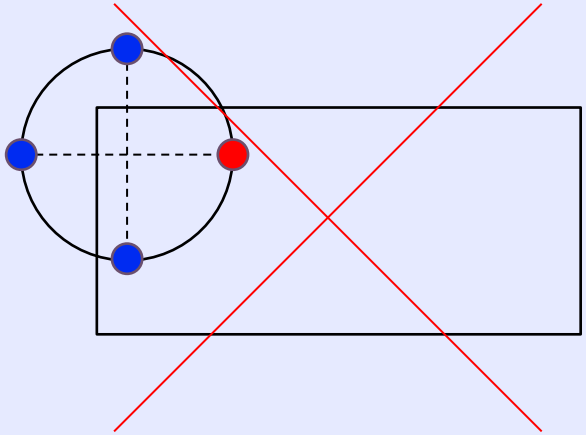
としておけば、それらしく見える。

当たり判定

- 左から入り込んだ場合を考える。





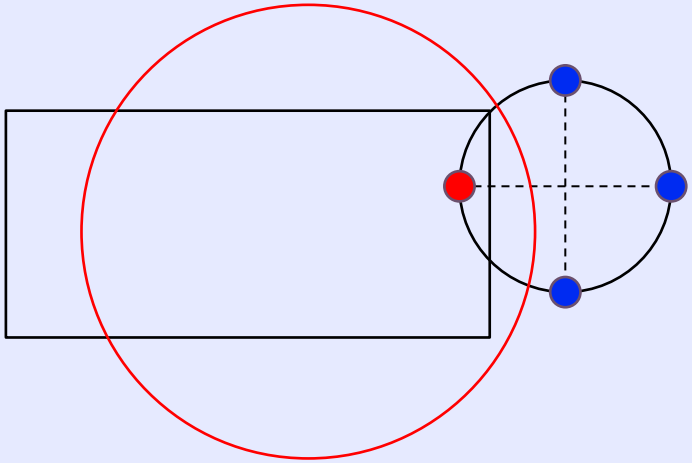


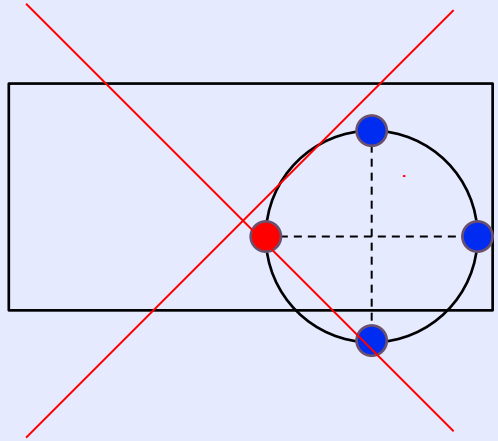
赤い点のみが入り込んでいる場合、○

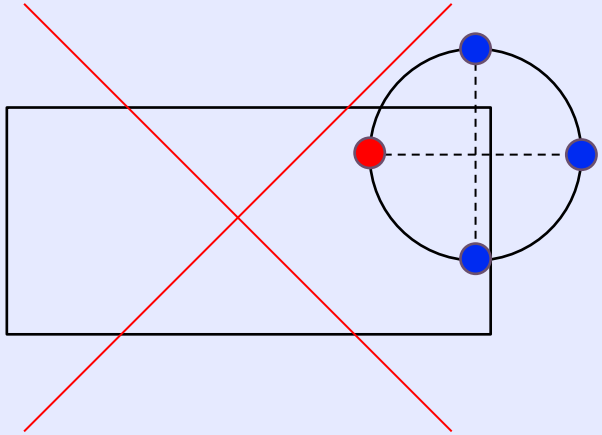
青い点も入り込んでいる場合、×

当たり判定

- 右から入り込んだ場合を考える。





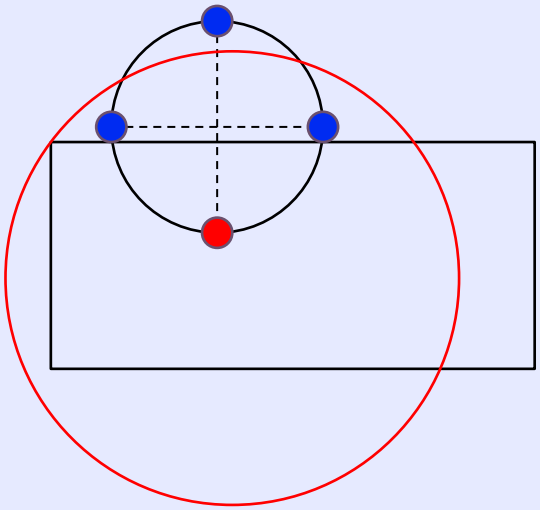


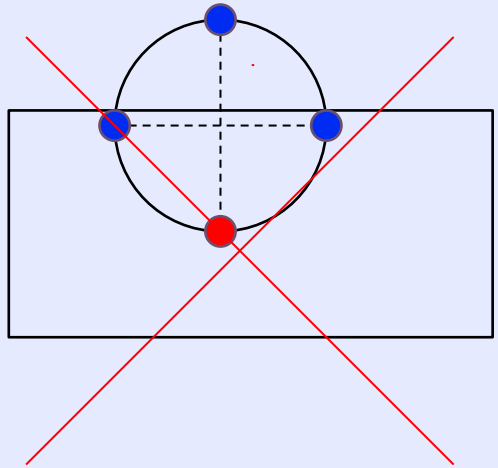
赤い点のみが入り込んでいる場合、○

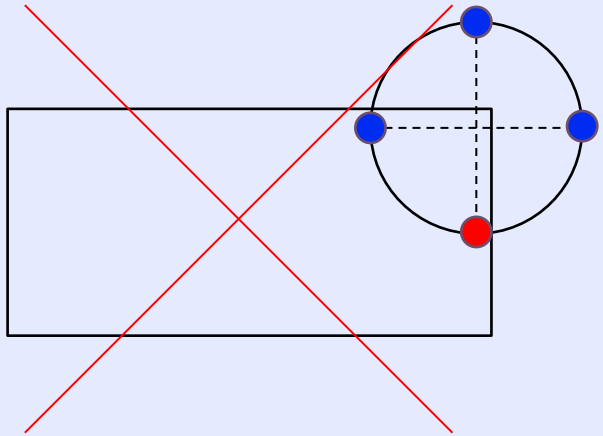
青い点も入り込んでいる場合、×

当たり判定

- 上から入り込んだ場合を考える。





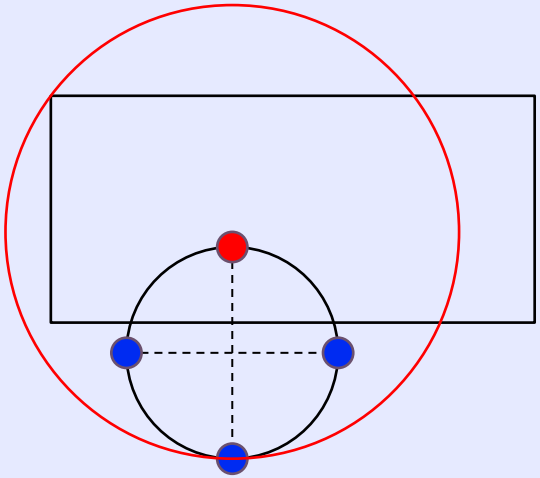


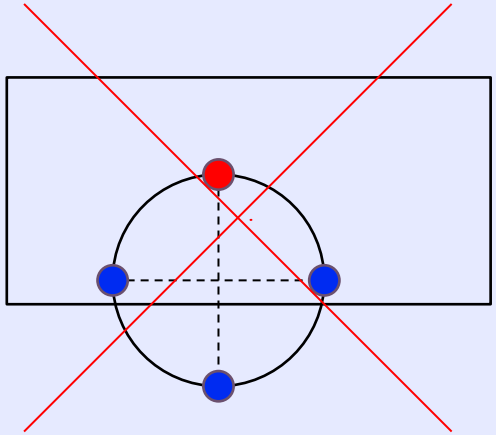
赤い点のみが入り込んでいる場合、○

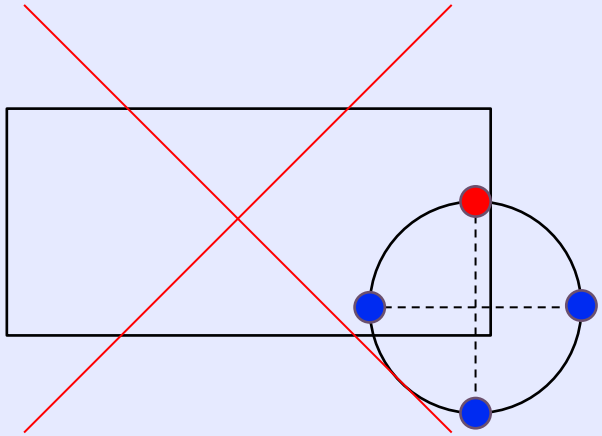
青い点も入り込んでいる場合、×

当たり判定

- 下から入り込んだ場合を考える。







赤い点のみが入り込んでいる場合、○

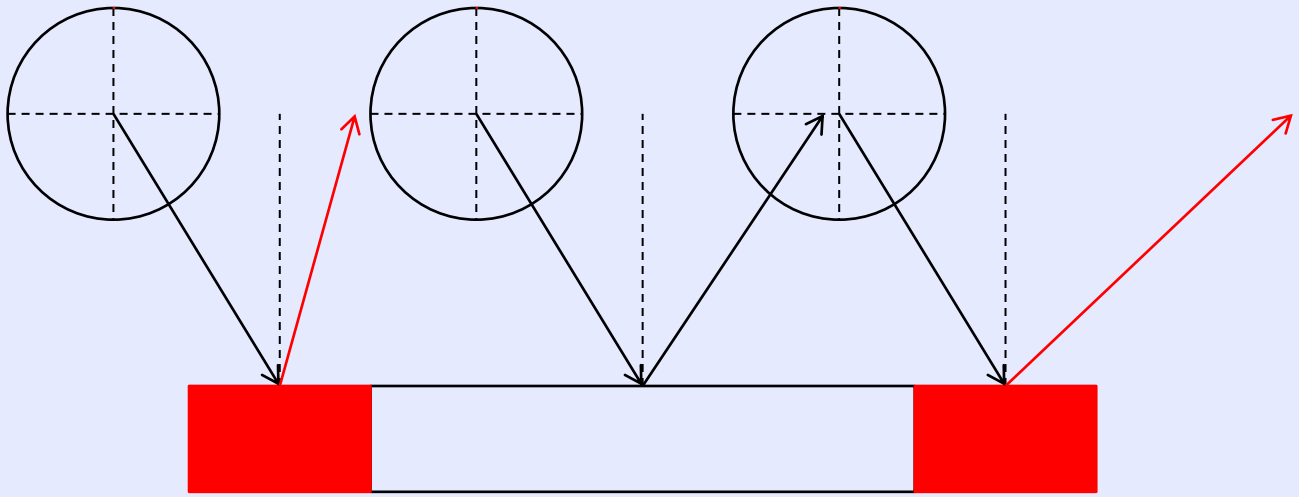
青い点も入り込んでいる場合、×

当たり判定

- バーとボールの当たり判定を考える
- 長方形と円の当たり判定であるので、ブロックの場合もほぼ同じ。
- 次項からはバーとボールの当たり判定を考える。

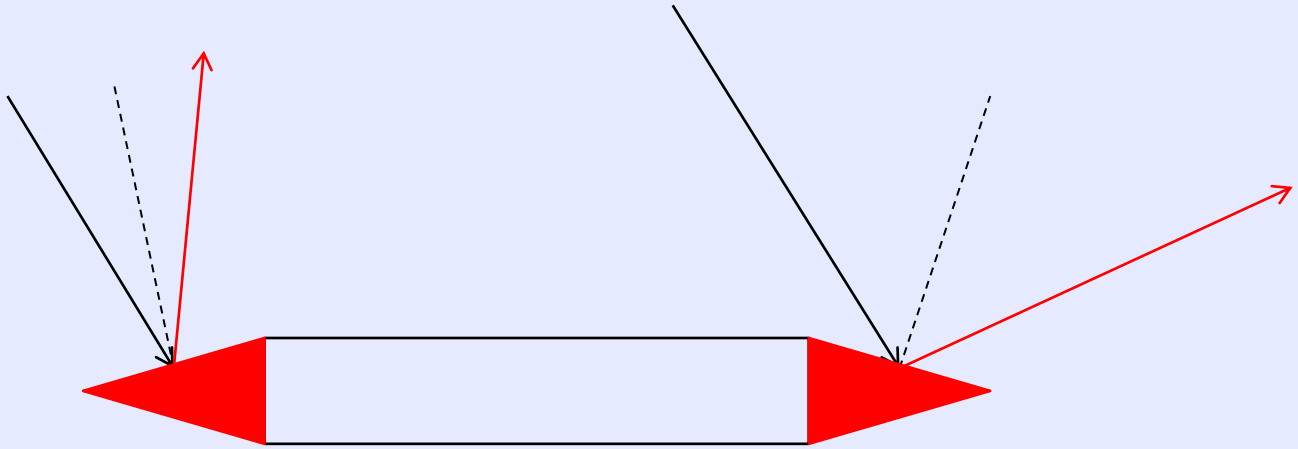
当たり判定

- ブロック崩しのバーの当たり判定は下図のようになる。
- ボールの反射パターンが一定にならないようにするため



当たり判定

- こんな感じのバーと考えて反射させる



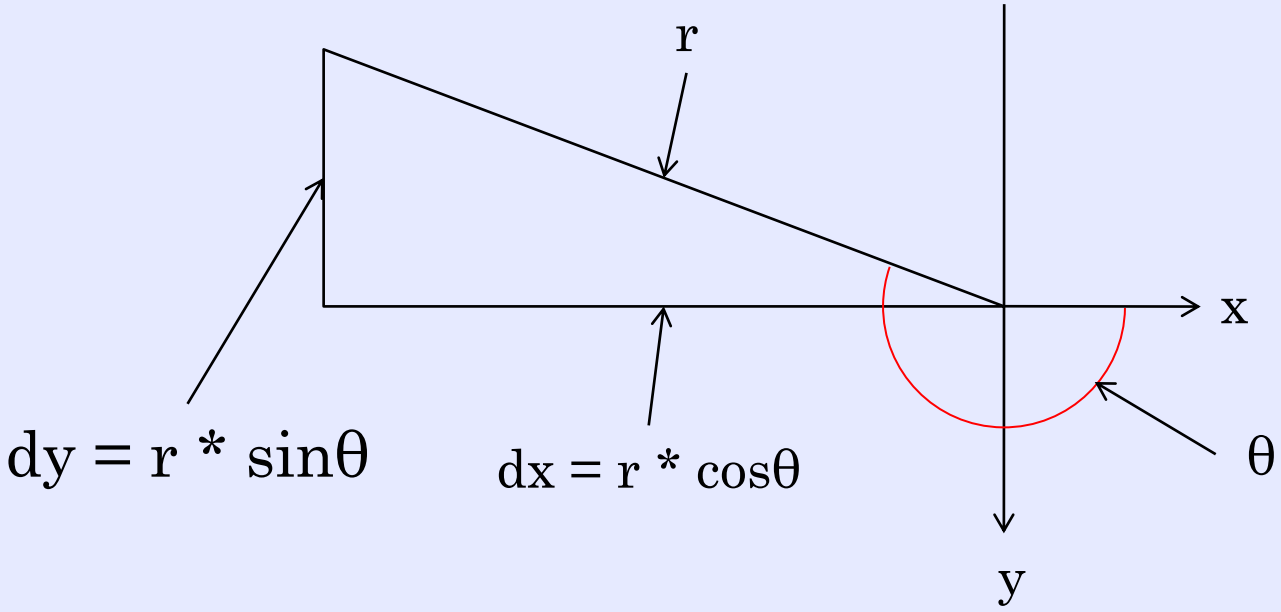
当たり判定

- 入射角を225°とすると、反射角は315°である。
  しかし、前項の左側の赤い部分の場合は反射角は、
  　　315° － 20° ＝ 295°
  右側の赤い部分の場合は反射角は、
  　　315° ＋ 20° ＝ 335°
  くらいにするといいかな。

当たり判定

- 角度からdx, dyを求める

当たり判定



- r = Math.sqrt(dx * dx + dy * dy);

当たり判定

- 反射前のdx, dyで、
  dx = -dx;
  とすると、通常の反射後のdx, dyが求まる。

- バーの左側部分の場合、
  1. そのdx, dyから角度を求め、
  2. その角度に-20する。
  3. そこから、またdx, dyを求めれば良い。
- バーの右側部分の場合、+20する。

当たり判定

```
double r = Math.sqrt(dx * dx + dy * dy);
dx = -dx;
double radian = Math.atan2(dy, dx);
double angle = Math.toDegrees(radian) - 20;
if(180 <= angle)          angle -= 360;
else if(angle <= -180)    angel += 360;
radian = Math.toRadians(angle);
dx = Math.cos(radian) * r;
dy = Math.sin(radian) * r;
```

- 上の例はボールがバーの左側部分に当たったと判定した時に反射後のdx, dyを求めるコードである。
- Math.atan2(y, x)はy / xの傾きの角度を-PI~PIの間で返す。
- Math.toDegrees(radian)は-PI~PIのradianを°に変換した値を返す。
- Math.toRadians(angle)は-180~180のangleをラジアンに変換した値を返す。

※PI……パイ

実行クラスを完成させる

当たり判定

- したがって、左側部分、中央部分、右側部分に分けて判定を行うことになる。

- reflect()メソッドに記述してみよう。

各画面を作成する

初期画面(scene = 0)

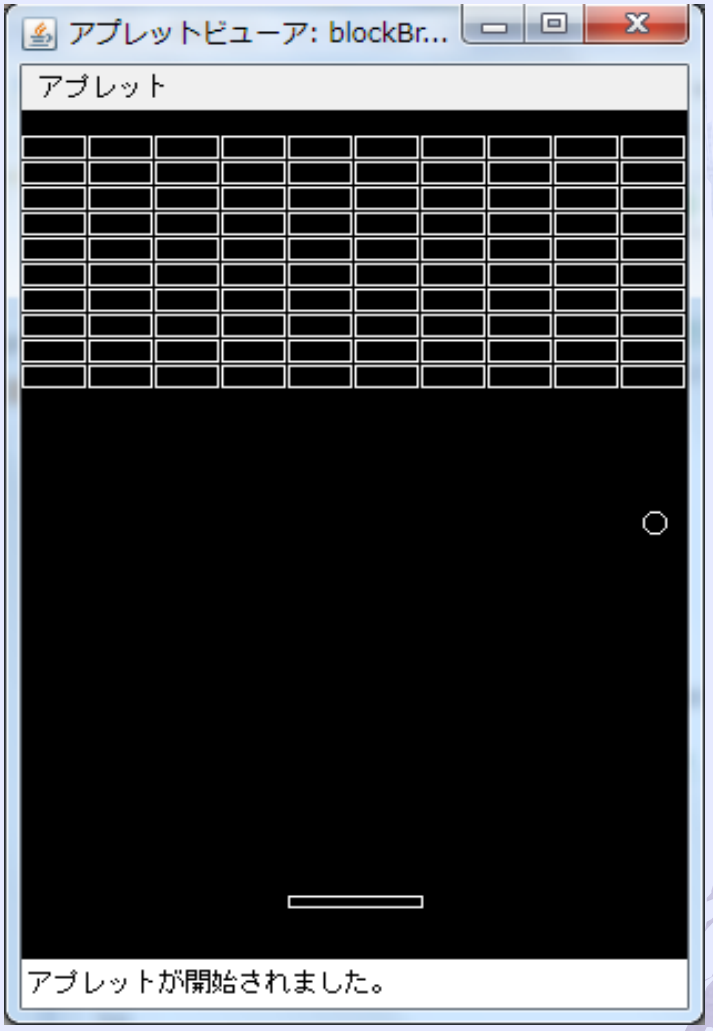
- 初期画面はこんな感じ。
- ready()メソッドに記述する
- この画面でスペースキーが押された場合、scene = 1にする。

プレイ中画面(scene = 1)

- プレイ中画面は、ボール、バー、ブロックを描画する。
- playing()メソッドに記述する

- ボールが落ちたら、scene = 2にする

- ブロックが全て破壊された場合、scene = 3にする



ゲームオーバー画面(scene = 2)

- プレイタイムは無くても良い
- gameover()メソッドに記述する

- この画面でスペースキーが押された場合、scene = 0にする

ゲームクリア画面(scene = 3)

- クリアタイムは無くても良い
- gameclear()メソッドに記述する
- この画面でスペースキーが押された場合、scene = 0にする

各画面を作成する

各画面の作成の流れとしては、

1. 描画色を指定する
   (例)g.setColor(Color.WHITE);
2. 描画する文字のフォントを指定する
   (例)Font f = new Font(Font.DIALOG, Font.ITALIC, 40);
   g.setFont(f);
3. 文字等を描画する

次項以降で画面作成に必要と思われるメソッドやその使い方を紹介していく。

g.setColor()について

- Graphicsクラスのset Color()メソッドは引数にColorオブジェクトを渡さなければならない
- Colorオブジェクトは、
「Color.色名」
「new Color(r, g, b)」
「new Color(r, g, b, a)」
　　(aは0~255で透明度を表す)
以上の取得方法がある。

g.setFont()について

- Graphicsクラスの setFont()メソッドは引数にFontオブジェクトを渡さなければならない。
- Fontオブジェクトは、
「new Font(フォント名, スタイル名, サイズ)」
で取得できる。
- フォント名は
Font.DIALOG, Font.DIALOG_INPUT, Font.MONOSPACED, Font.SANS_SERIF, Font.SERIFのうちいずれか。
- スタイル名は
Font.PLAIN, Font.BOLD, Font.ITALICのうちいずれか。
- サイズは文字の大きさ(整数値)を表す。

画像を描画する

- g.drawImage(Imageオブジェクト, x, y,
  width, height, null);
  drawRect()メソッドとほぼ同じで、四角形の領域にImageオブジェクト(画像)を描画する。

Imageクラス

- 画像を扱うクラスとして、JavaにはImageクラスというものがある。
- Imageクラスを使うと、前項のようにGraphicsクラスのdrawImage()メソッドで簡単に画像が描画できる。

- Image image =
  new ImageIcon(“ファイルパス”).getImage();
  とすることでImageオブジェクトを取得できる。
- ファイルパスは相対パスで指定したほうが良い。
  画像をプロジェクトの直下に保存した場合、
  ファイルパスは、「../画像名」となる。

キーボード入力を受け取る

KeyListenerを登録する

- init()メソッドに、
  addKeyListener(this);
  という記述を追加する。
  これはKeyListenerを実装したクラスを引数として渡すことでキーリスナーを登録できる。
  するとキーが押された時、keyPressed()メソッドが呼ばれるようになる。

- 試しに、keyPressed()メソッドに
  systrace →[Ctrl+Space]で出力を追加して、呼び出されているかどうか確かめてみよう。
  ※最初にアプレットのウィンドウをクリックして、フォーカスを合わせてから、キーを押す必要がある。

KeyEventを受け取る

- keyPressed()メソッドはKeyEventオブジェクトを引数として受け取っている。
- KeyEventオブジェクトには、どのキーが押されたか等の情報が格納されている。

- 次項にkeyPressed()メソッドの例を示す

KeyEventを受け取る

```
public void keyPressed(KeyEvent e){
        int keyCode = e.getKeyCode();
        switch(keyCode){
        case KeyEvent.VK_LEFT:
                break;
        case KeyEvent.VK_RIGHT:
                break;
        case KeyEvent.VK_SPACE:
                break;
        }
}
```

- こんな感じで分岐できる。(if文でもOK)
- e.getKeyCode()で取得した整数値とKeyEvent.VK_キーの名前を比べることでどのキーが押されたかを判別することになる。

KeyEventを受け取る

- 前項でkeyPressed()メソッドの雛形を示したが、具体的な処理は自分で考えてみよう。

- keyPressed()メソッドでやらなければならないことは、
  スペースボタンが押されたとき、
  　　scene = 0ならば、scene = 1
  　　scene = 2 または scene = 3ならば、
  　　scene = 0
  左矢印キーが押されたとき、
  　　scene = 1ならば、barを左へ移動させるための処理
  右矢印キーが押されたとき、
  　　scene = 1ならば、barを右へ移動させるための処理

プレイ中画面からの画面切り替え

- 「プレイ中画面(scene = 1)」からは
  「ゲームオーバー画面(scene = 2)」
  「ゲームクリア画面(scene = 3)」
  へそれぞれ画面が切り替わる。

- 「ゲームオーバー画面(scene = 2)」
  ボールが落ちたとき
- 「ゲームクリア画面(scene = 3)」
  ブロックが全て破壊されたとき

- これらを実装しなければならない

プレイ中画面からの画面切り替え

まずは、自分でどこに記述すればよいかを考えてみよう。

プレイ中画面からの画面切り替え

終わり