オブジェクト指向プログラミング(2)

第6回

横山　孝典
E-mail: tyoko@tcu.ac.jp

Java言語に関する補足事項

- 例外処理

例外処理(1)

Javaが提供している例外処理(エラー処理)機構

```
try {
    例外を起こすかもしれないコード
} catch(Exception e) {
    例外を処理するコード
}
```

例外処理(2)

例外処理の例

ソースコード

```
// ExTest
public class ExTest {
  int x = 1;
  int y = 1;

  public static void main(String[] args) {
    (new ExTest()).run();
  }

  public void run() {
    try {
      y--;
      x = x / y;
    } catch(ArithmeticException e) {
      System.out.println(e.getMessage());
      x = 1;
    }
    System.out.println("(x, y) = (”
                    + x + ", " + y + ")");
  }
}
```

実行結果

```
>javac ExTest.java

>java ExTest

/ by zero
(x, y) = (1, 0)
```

参考:文字列入力(1)

文字列入力方法の例

```
import java.io.*;   // 入出力(IO)に関するクラスを利用する
・・・・
public class ・・・・{
・・・・
  public ・・・・{
    ・・・・
    String inStr;
    BufferedReader in =
       new BufferedReader(new InputStreamReader(System.in));
    try{
      System.out.print("入力してください:");
      inStr = in.readLine();    // 文字列入力
    } catch(IOException e) {
      System.out.println(e);    // 例外(IOException)を表示
  }
    ・・・・
  }
・・・・
}
```

注)例外処理がないとコンパイルエラーになる

参考：文字列入力（2）

簡便な方法（例外を指定するのみでcatchしない）

```
import java.io .*;   // 入出力（IO）に関するクラスを利用する
・・・・
public class ・・・・{
・・・・
  public ・・・・ throws IOException {
    ・・・・
    String inStr;
    BufferedReader in =
       new BufferedReader(new InputStreamReader(System.in));
    System.out.print("入力してください:");
    inStr = in.readLine();    // 文字列入力
    ・・・・
  }
・・・・
}
```

注）例外処理がないとコンパイルエラーになる

オブジェクト指向設計の指針

- システムが何をするかを考える前に、
  何をオブジェクト（クラス）にするかを考える
  - 手続きを考える前に、データに注目する
    - オブジェクトは「機能」ではない
  - 順序（処理の順番）を考えるのはできるだけ後にする

クラスの抽出

- 対象とする世界（問題）に存在する「モノ」をクラスにする
  - 実際には、モノそのものではなく、それを計算機上の表現に変えたもの（モデル）
  - クラスは「作る」のではなく「拾う」

- 「名詞」に着目する
  - ただし、全ての名詞がクラスに対応するわけではない
    - 属性や値を表す名詞もある
    - プログラムの対象外の名詞が含まれることもある
  - 属性や状態を持ち、意味のある操作（メソッド）によって特徴付けられるもののみがオブジェクト（クラス）になりうる

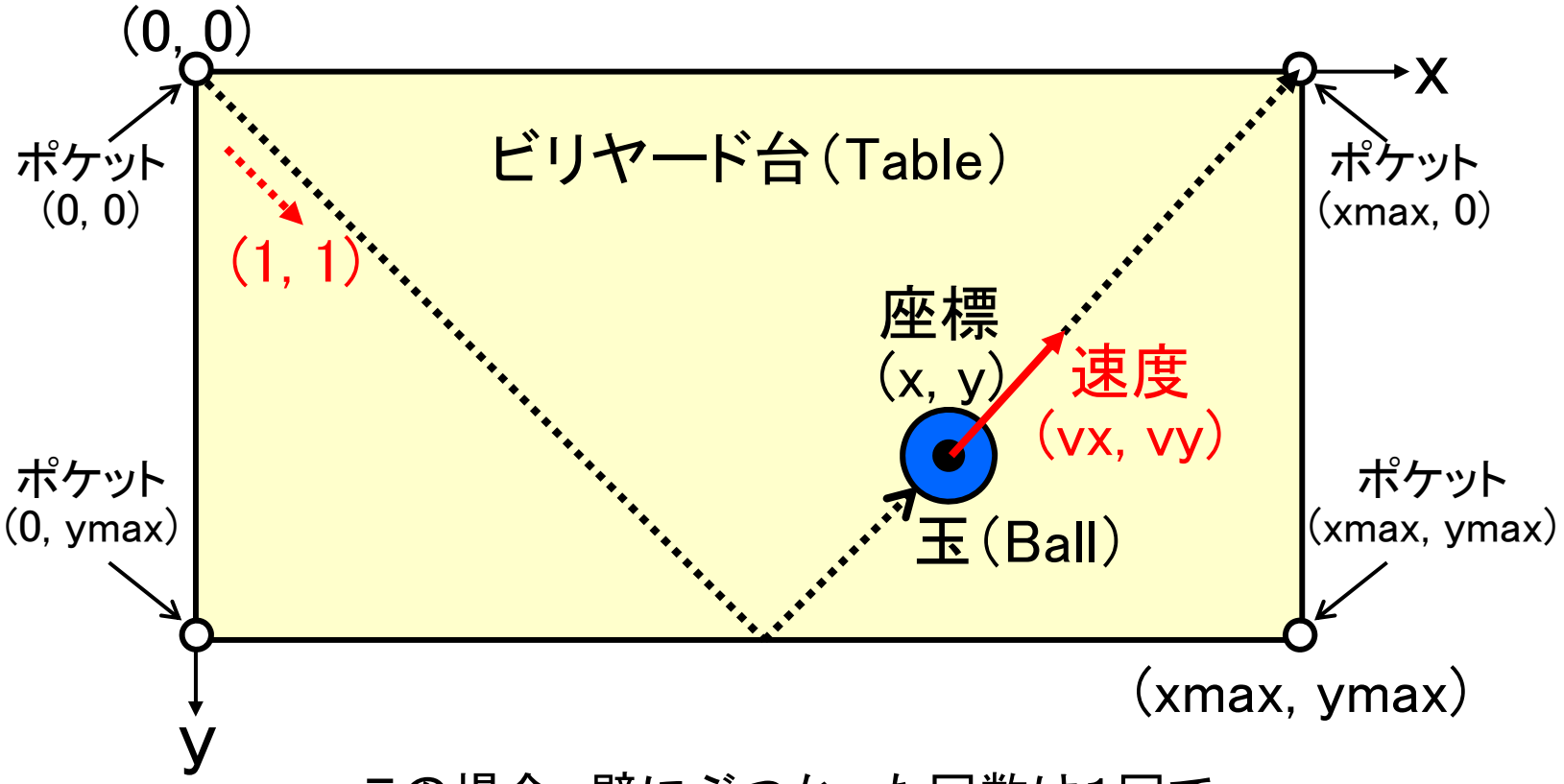
例題（1）

「ビリヤード」プログラム

ビリヤードのプログラムを作成する。
ビリヤード台の大きさは任意（xmax, ymax）に決めることができるものとする。
ポケットの位置は，４角（0, 0）（xmax, 0）（0, ymax）（xmax, ymax）のみである。
玉のスタートする位置は左上で，右下45度に向けて速度（1, 1）で玉を転がす（摩擦は考慮しない）。
玉の大きさは考慮しなくてよい。
玉を画面に表示するともに、最後に、何回壁にぶつかって，どのポケットに落ちたかを示す。

例題（2）

「ビリヤード」プログラム



この場合、壁にぶつかった回数は1回で、
ポケット(xmax, 0)に落ちる。

例題のクラス抽出（1）

名詞を抽出すると

ビリヤードのプログラムを作成する。
ビリヤード台の大きさは任意（xmax, ymax）に決めることができるものとする
ポケットの位置は，４角（0, 0）（xmax, 0）（0, ymax）（xmax, ymax）のみである
玉のスタートする位置は左上で，右下45度に向けて速度（1, 1）で玉を転がす（摩擦は考慮しない）。
玉の大きさは考慮しなくてよい。
玉を画面に表示するともに、最後に、壁にぶつかた回数と，どのポケットに落ちたかを示す。

ビリヤード、ビリヤード台、大きさ、ポケット、位置、角、玉、左上、右下、速度、摩擦、画面、壁、回数

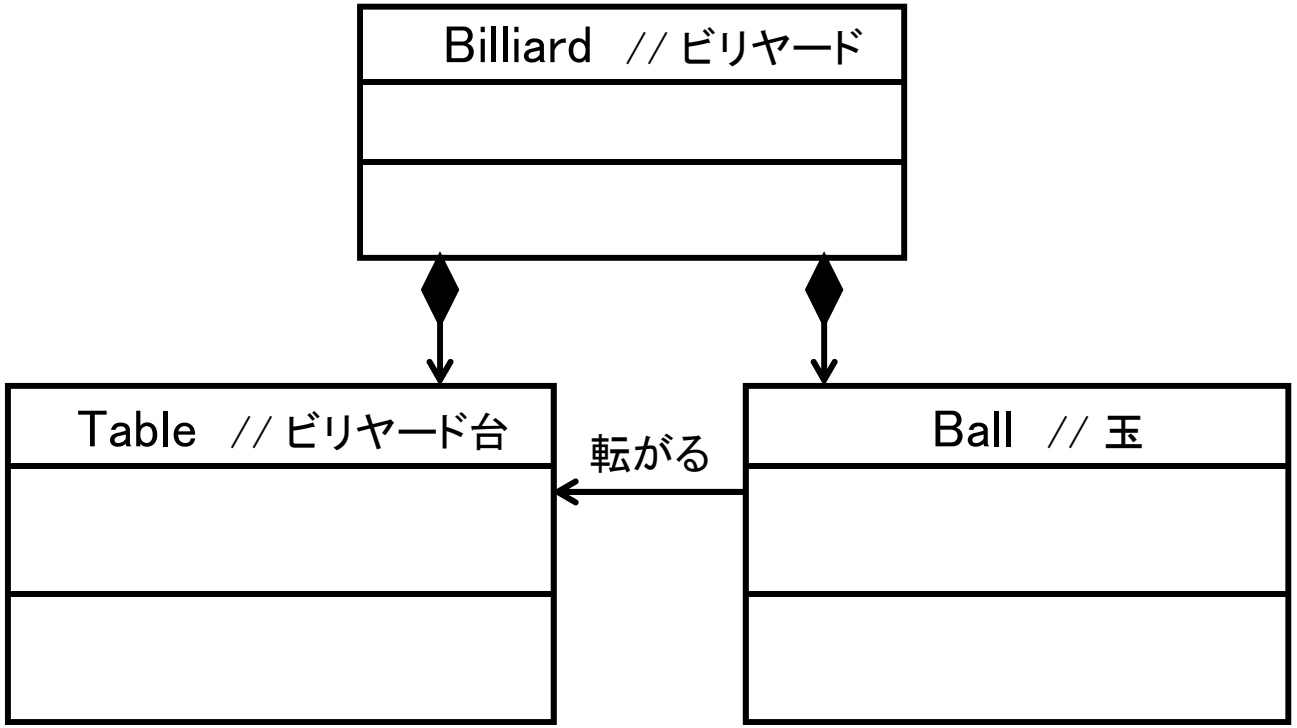
例題のクラス抽出（2）

- クラスとするもの
  - ビリヤード：対象とするゲームそのものなのでクラスとする
  - ビリヤード台：対象問題に存在する「モノ」であり、クラスとする
  - 玉：対象問題に存在する「モノ」であり、クラスとする
  - 画面：表示に関する「モノ」でありクラスとする
    ただし、今回は与えられるものとし、設計対象とはしない

- クラスとしないもの
  - 大きさ：「ビリヤード台」の属性である
  - ポケット、壁：対象問題に存在する「モノ」（ビリヤード台の部分）ではあるが、座標で表現できるのでクラスとはしない
  - 位置、速度、回数：「玉」の属性である（位置は座標）
  - 角、左上、右下：位置（座標）の値である
  - 摩擦：考慮しなくてよい（対象外）

例題のクラス抽出（3）

- 演習問題のクラス構成（画面表示の機能は除く）
  - ビリヤード台（Table）、玉（Ball）は、ビリヤード（Billiard）を構成する部品
  - 玉（Ball）はビリヤード台（Table）を転がる



クラスの設計

オブジェクトの属性の抽出

属性とは？

- オブジェクトの持つ情報
- オブジェクトの状態

属性は隠蔽するのが原則

- private または protected とする

オブジェクトの振る舞いの設計

各オブジェクトは自分に責任を持つ

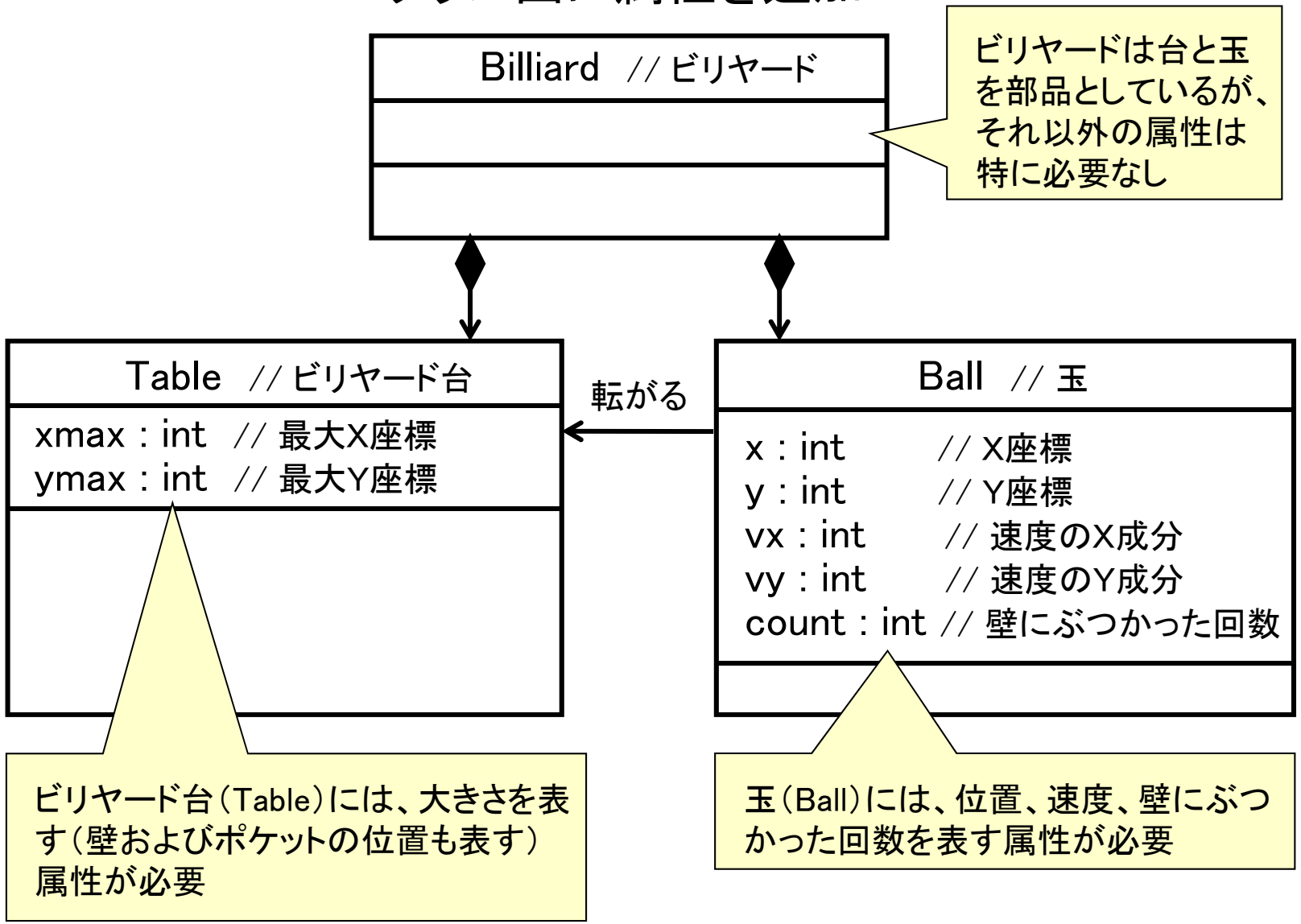
- 自分のことは自分でする
- 自分の属性に対する処理は自分で行う

他のオブジェクトには処理させない

→ “set属性()” というメソッドはできる限り作らない

例題におけるオブジェクトの属性

クラス図に属性を追加



設計のポイント（1）

- クラス設計のポイント
  - 再利用を考慮した設計
    - 様々な使い方がされる可能性を考慮
  - 拡張性を考慮した設計
    - 継承を利用しやすいクラス設計
  - 保守性を考慮した設計
    - 独立性の高いクラスにする（他のクラスに依存しない）

設計のポイント(2)

- インタフェース設計のポイント
  - 少ないインタフェース
    - 他のオブジェクトとのやりとり(メソッド呼び出し)をできるだけ少なくする
  - 小さいインタフェース(弱い結びつき)
    - オブジェクト間でやりとりする情報(引数、戻り値)をできるだけ少なくする
  - 明示的なインタフェース
    - オブジェクト間のやりとり(メソッド呼び出し)は、呼び出し側および呼び出される側にとって、明らかになっていなければならない

- メソッド設計のポイント
  - 単純で明確な動作
    - 1つのメソッドは1つの仕事だけをする

演習問題その1（1）

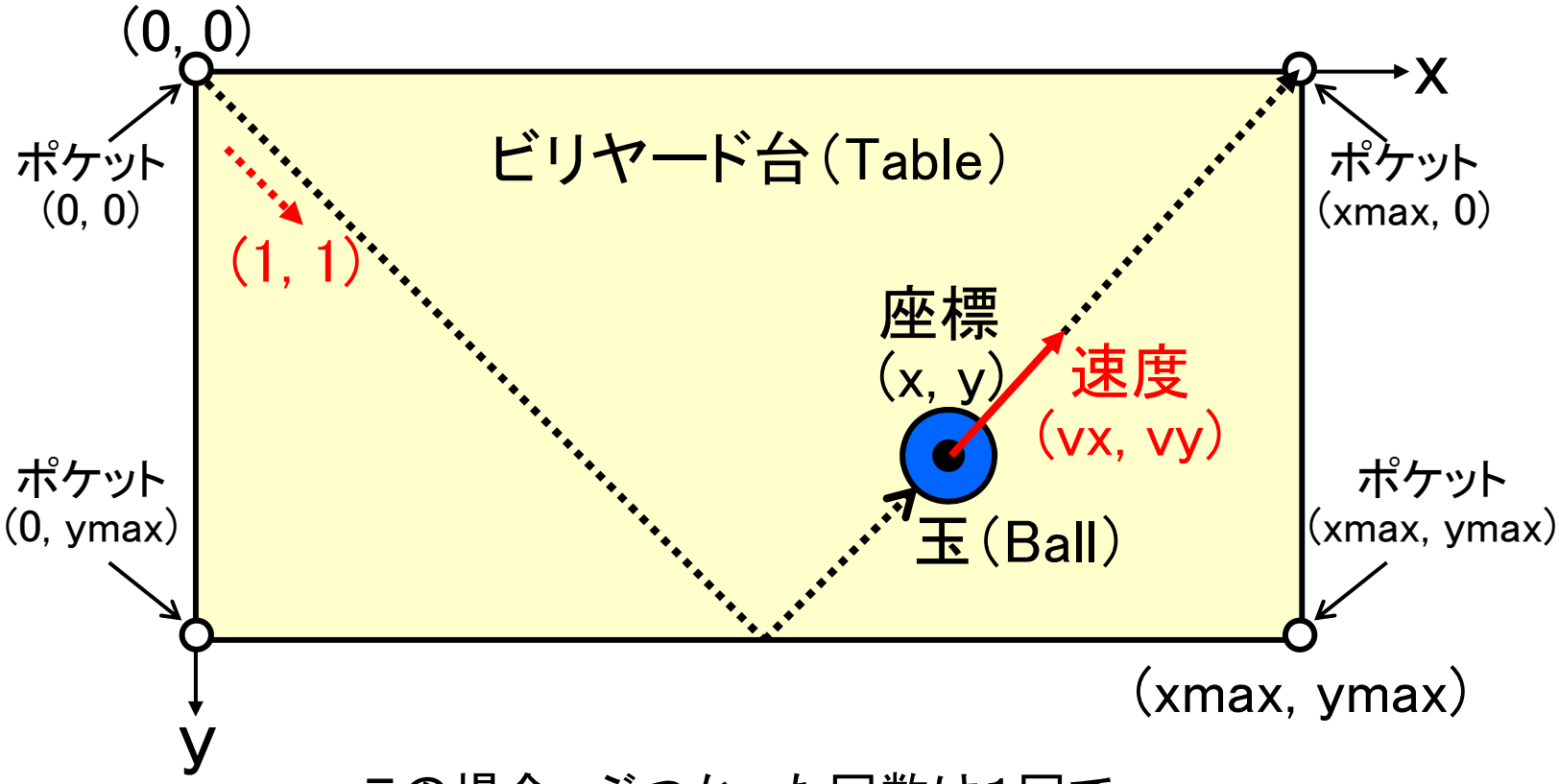
「ビリヤード」のプログラムを作成する（画面表示の機能も実現する）。

ビリヤードのプログラムを作成する。
ビリヤード台の大きさは任意 (xmax, ymax) に決めることができるものとする。
ポケットの位置は，４角 (0, 0) (xmax, 0) (0, ymax) (xmax, ymax) のみである。
玉のスタートする位置は左上で，右下45度に向けて速度（1, 1）で玉を転がす（摩擦は考慮しない）。
玉の大きさは考慮しなくてよい。
玉を画面に表示するともに、最後に、何回壁にぶつかって，どのポケットに落ちたかを示す。

ビリヤードのクラス Billiard、ビリヤードインタフェース BilIF、ウィンドウを表示するためのクラス BilFrame は与える。
ビリヤード台のクラス Table、玉のクラス Ball を作成する。

演習問題その1（2）

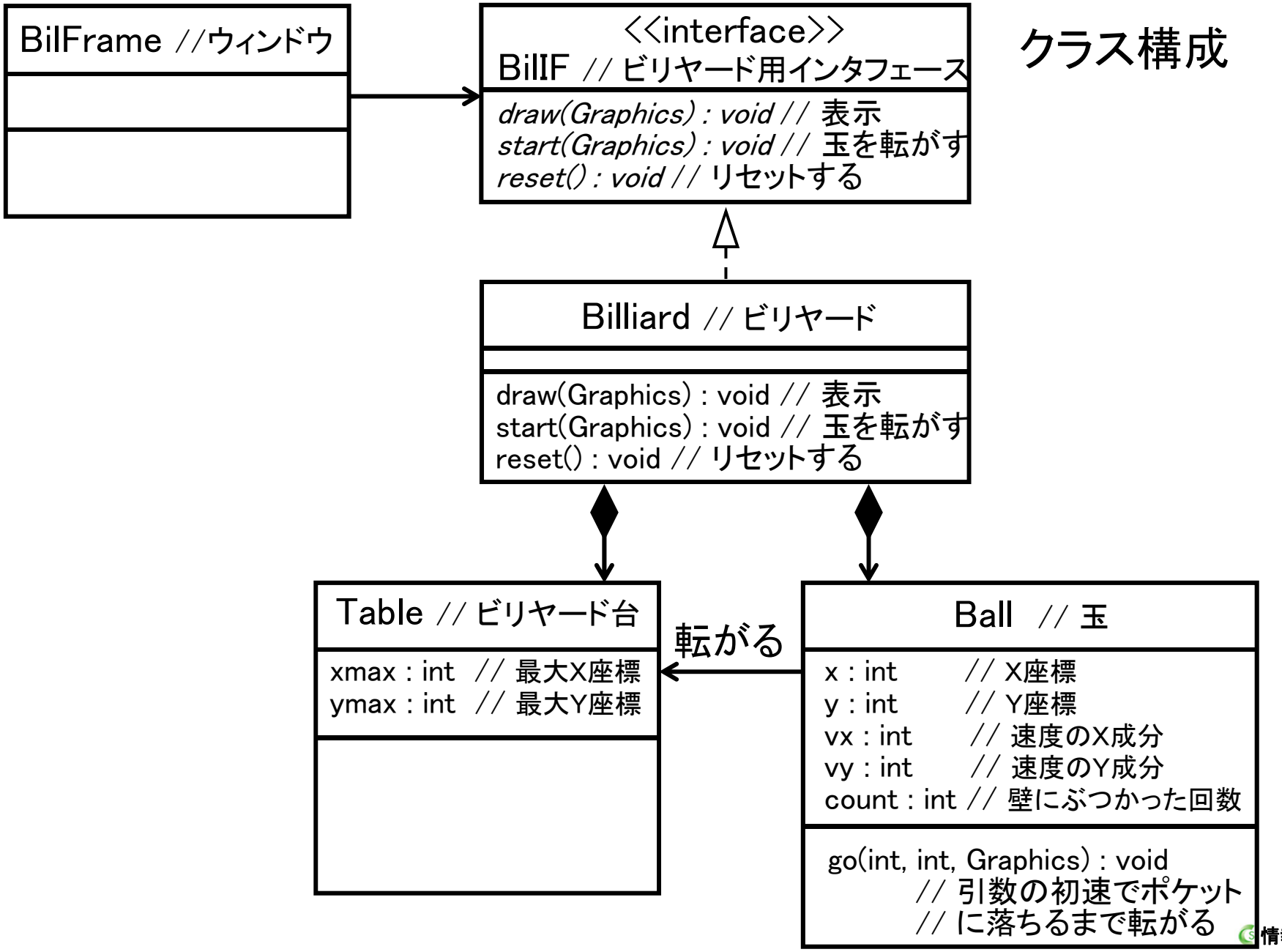
「ビリヤード」プログラム



この場合、ぶつかった回数は1回で、
ポケット(xmax, 0)に落ちる。

演習問題その1(3)

クラス構成



演習問題その1（4）

Billiardの動作

プログラムの起動法は下記。

```
>java Billiard
ビリヤード台の横方向の大きさは？ 800
ビリヤード台の縦方向の大きさは？ 370
```

ビリヤード台（Table）の大きさをキーボードから入力する

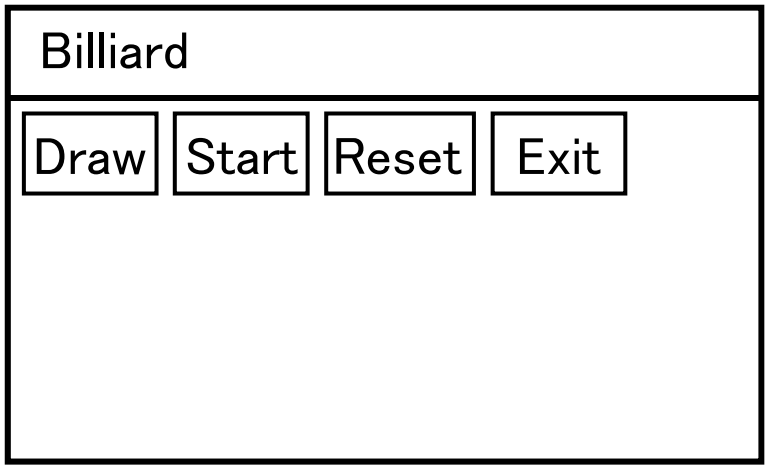
右のようなウィンドウが現れる。

① ボタン“Draw”をクリックすると、
Biliard のメソッド draw() が呼び出される。
（ビリヤード台と玉を表示する）

② ボタン“Start”をクリックすると、
Biliard のメソッド start() が呼び出される。
（玉が転がりだす）

③ ボタン“Reset”をクリックすると、
Biliard のメソッド reset() が呼び出される。
（玉の座標とぶつかった回数をリセットする）

④ ボタン“Exit”をクリックすると、
ウィンドウが消えて終了。



注）③の後、①②を行うことで、
何度も実行可能

演習問題その1（5）

Billiardの動作

ソースコードを与えるので、よく読んで理解すること。
　注）BilFrame については読まなくてもよい。後掲するシーケンス図をよく理解する。
メソッド start() の最後で下記のような結果が表示される。

ボールがポケット(800, 0)に落ちました。
壁に115回ぶつかりました。

以下の場合の表示結果
xmax = 800
ymax = 370

Billiardのソースコード（1）

```
// Billiard.java

import java.io.*;            // 入出力(IO)に関するクラスを利用する
import java.awt.Graphics;    // クラスGraphicsを利用する
import java.awt.Color;       // クラスColorを利用する

public class Billiard implements BilIF {   // ビリヤードのクラス
                                           // ビリヤードインタフェースを継承
  Table table;   // ビリヤード台のインスタンスの参照
  Ball ball;     // 玉のインスタンスの参照
```

演習問題その1（6）

Billiardのソースコード（2）

```
 // メインプログラム
public static void main(String[] args) {
  String inStr;  // 入力文字列記憶用
  BufferedReader in =
    new BufferedReader(new InputStreamReader(System.in)); // 文字列入力
  try{
    System.out.print("ビリヤード台の横方向の大きさは？ ");
    inStr = in.readLine();                  // 文字列入力
    int m = Integer.parseInt(inStr);        // 文字列を整数に変換
    System.out.print("ビリヤード台の縦方向の大きさは？ ");
    inStr = in.readLine();                  // 文字列入力
    int n = Integer.parseInt(inStr);        // 文字列を整数に変換
    Billiard bil = new Billiard(m, n);  // ビリヤードのインスタンス生成
    new BilFrame("Billiard", 200 + m, 200 + n, bil);
                          // ビリヤード用フレーム(ウィンドウ)のインスタンス生成
                          // 画面サイズはビリヤード台より上下左右100ずつ大きくする
  } catch(IOException e) {              // IO例外が発生した場合
    System.out.println(e.getMessage());
    System.out.println("もう一度やりなおしてください。");
  } catch(NumberFormatException e) {  // 数字フォーマット例外が発生した場合
    System.out.println(e.getMessage());
    System.out.println("もう一度やりなおしてください。");
  }
}
```

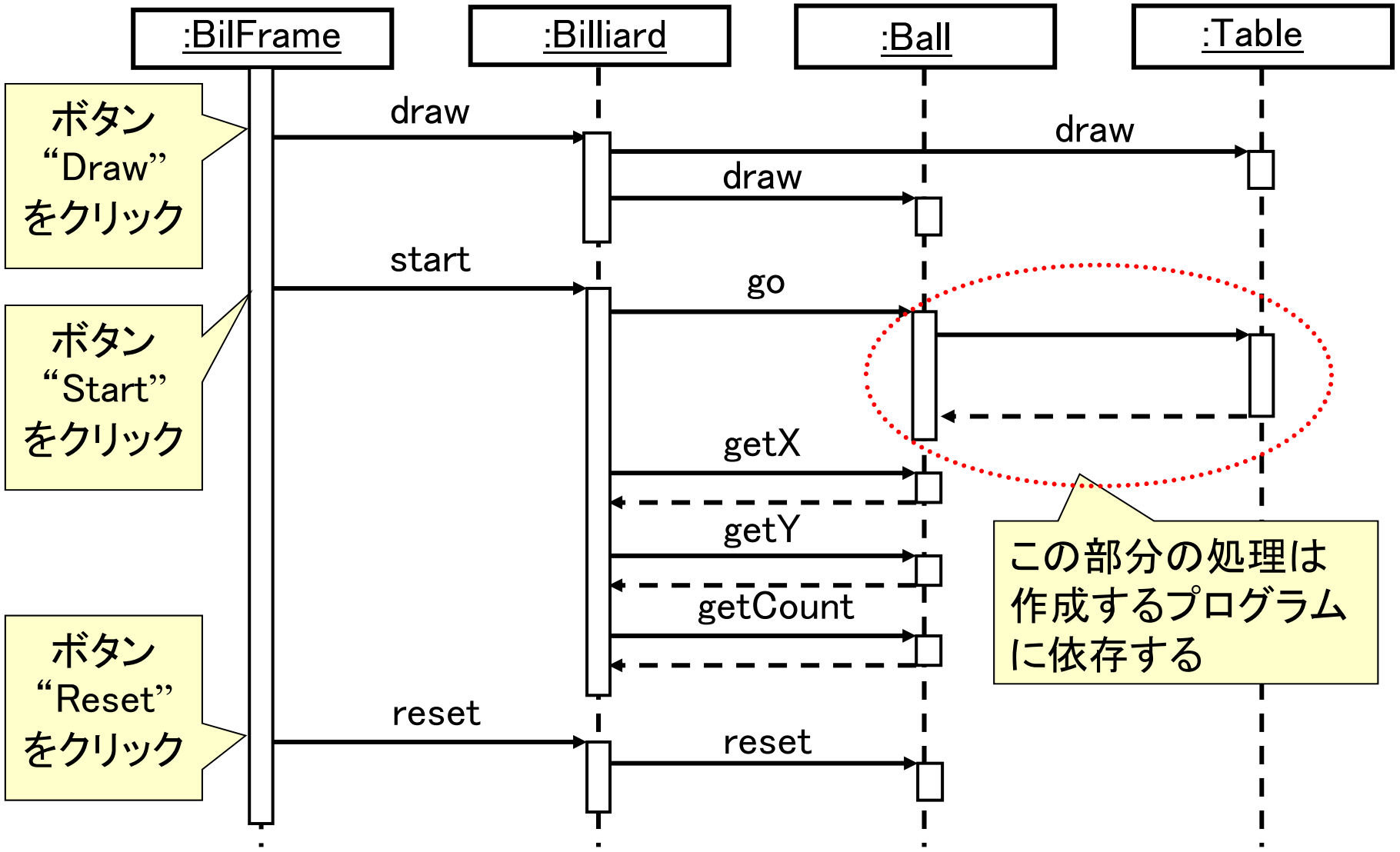
演習問題その1（7）

Billiardのソースコード（3）

```
// コンストラクタ
 public Billiard(int xmax, int ymax) {   // 引数はテーブルのサイズ
   table = new Table (xmax, ymax);   // ビリヤード台のインスタンス生成
   ball = new Ball(table, 0, 0);     // 玉のインスタンス生成(位置は左上端)
 }

 // ゲームの処理
 public void start(Graphics g) {   // 引数はGraphicsオブジェクト
   ball.go(1, 1, g);     // 玉を初速度(1, 1)で転がす
   System.out.println("玉がポケット(" + ball.getX() + ", "
                                  + ball.getY() + ")に落ちました。");
   System.out.println("壁に" + ball.getCount()
                                                  + "回ぶつかりました。");
 }

 // ビリヤードを表示するメソッド
 public void draw(Graphics g) {   // 引数はGraphicsオブジェクト
   table.draw(g);     // ビリヤード台を表示する
   ball.draw(g);      // 玉を表示する
 }

 // リセットするメソッド
 public void reset(Graphics g) {
   ball.reset();      // 玉をリセットする
 }

}
```

演習問題その1（8）

シーケンス図



演習問題その1(9)

ビリヤード台の表示方法

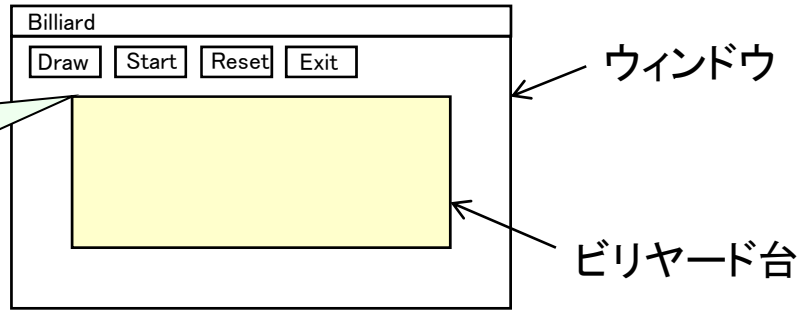
ソースファイルの先頭で下記を宣言する

```
import java.awt.Graphics;    // クラスGraphicsを利用する
import java.awt.Color;       // クラスColorを利用する
```

メソッド“draw()”により表示する

```
// ビリヤード台を表示するメソッド
public void draw(Graphics g) {    // 引数はGraphicsオブジェクト
  ? ? ? ? ? ? ;   // 描画する色を黄色にする
  ? ? ? ? ? ? ;   // 塗りつぶし矩形(ビリヤード台)を描画する
        // ウィンドウ上の座標(100,100)の位置をビリヤード座標の(0,0)とする
}
```

ウィンドウ上の座標(100,100)の位置を
ビリヤード台の座標の(0,0)とする



演習問題その1（10）

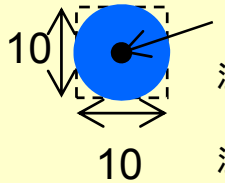
玉の表示方法

ソースファイルの先頭で下記を宣言する

```
import java.awt.Graphics;    // クラスGraphicsを利用する
import java.awt.Color;       // クラスColorを利用する
```

メソッド“draw()”により玉を表示する

```
// 玉を表示するメソッド
public void draw(Graphics g) {   // 引数はGraphicsオブジェクト
  ? ? ? ? ? ? ;   // 描画する色を青にする
  ? ? ? ? ? ? ;   // 塗りつぶし楕円(玉)を描画する
   // ウィンドウ上の座標(100,100)を
   // ビリヤード座標の(0,0)とする
   // 直径を10とする
}
```



注1）直径を10とするのは表示のためで、
運動については玉の大きさは考えなくてよい
注2）画面をクリアして再描画すると時間が
かかるので、玉の軌跡は残ったままでよい

メソッド“go()”中で、玉の表示を行う

```
public void go(int vvx, int vvy, Graphics g) {
  ・・・・
  do {
    ・・・・     // 玉を移動
    draw(g);    // 玉を表示
    ・・・・
  } while ・・・ // 玉がポケットに落ちるまで繰り返し
}
```

演習問題その1（11）

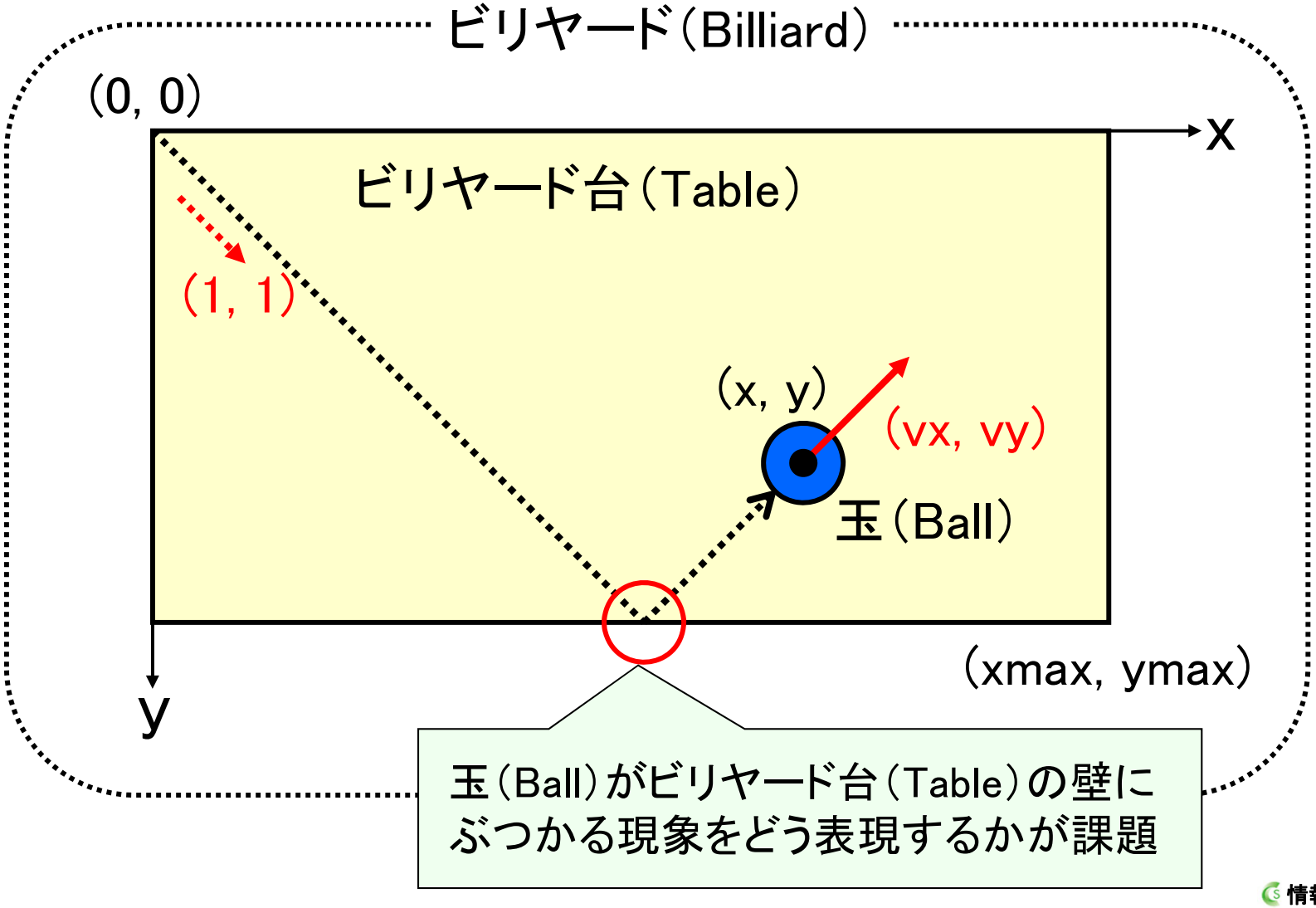
補足）
玉の動きを遅くしたい時には、Ballのメソッド“go()” に、以下のような、待ち時間を入れる処理を追加する。

```
public void go(int vvx, int vvy, Graphics g) {
  ・・・・
  do {
    try {
      Thread.currentThread().sleep(1);     // 1ms待つ
    } catch (InterruptedException e) {
      System.out.println(e.getMessage());
    }
     ・・・・
  } while (・・・);
 ・・・・
```

“sleep()” のかっこの中の数字（整数）を変更することで、待ち時間を変えることができる（ただし、大きくして遅くなりすぎないように注意）。

演習問題その1（12）

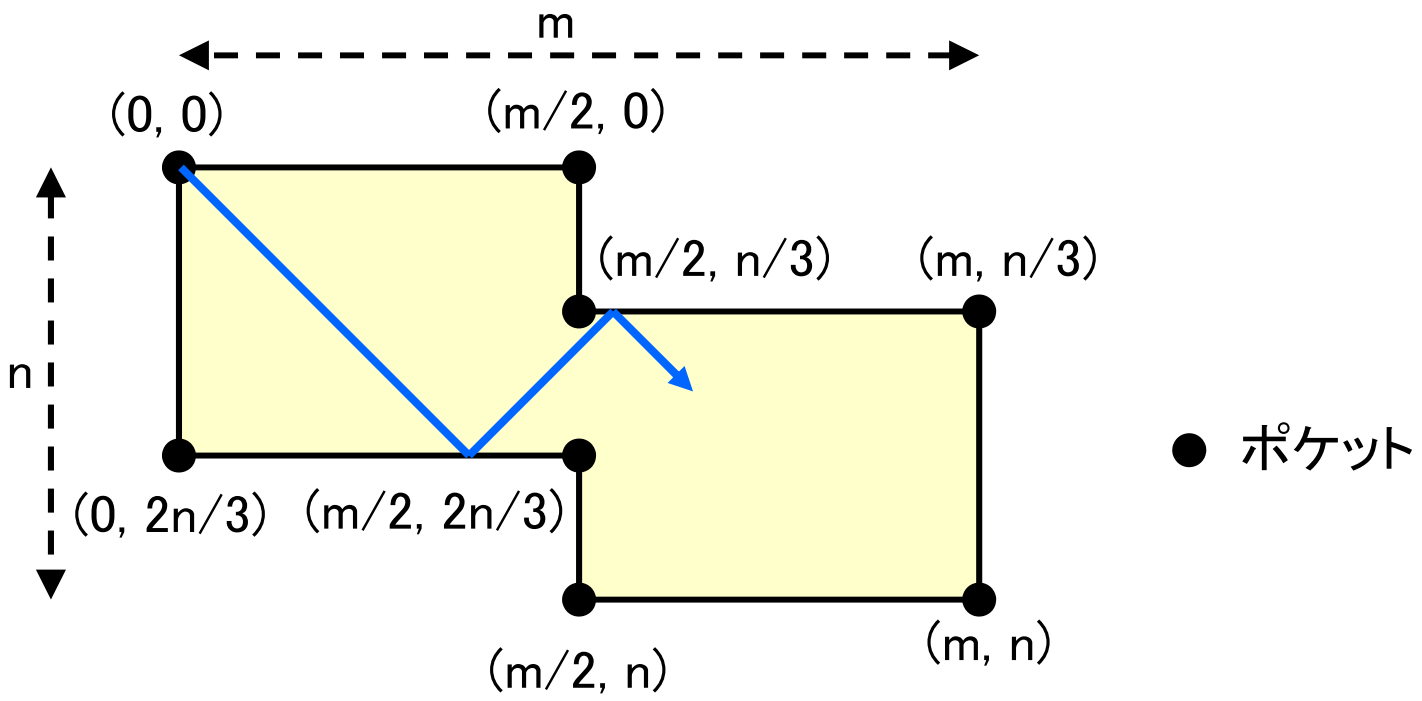
ポイント：各オブジェクトの振る舞いをよく分析する



演習問題その2（1）

演習問題その1の終了後、この「演習問題その2」を行う

下図のような変形ビリヤード台（StrangeTable）のクラスを作成し、通常のビリヤード台（Table）と入れ替えて、実行できるようにする。



注）この図では、演習問題その1の xmax は m、ymax は n と表現している

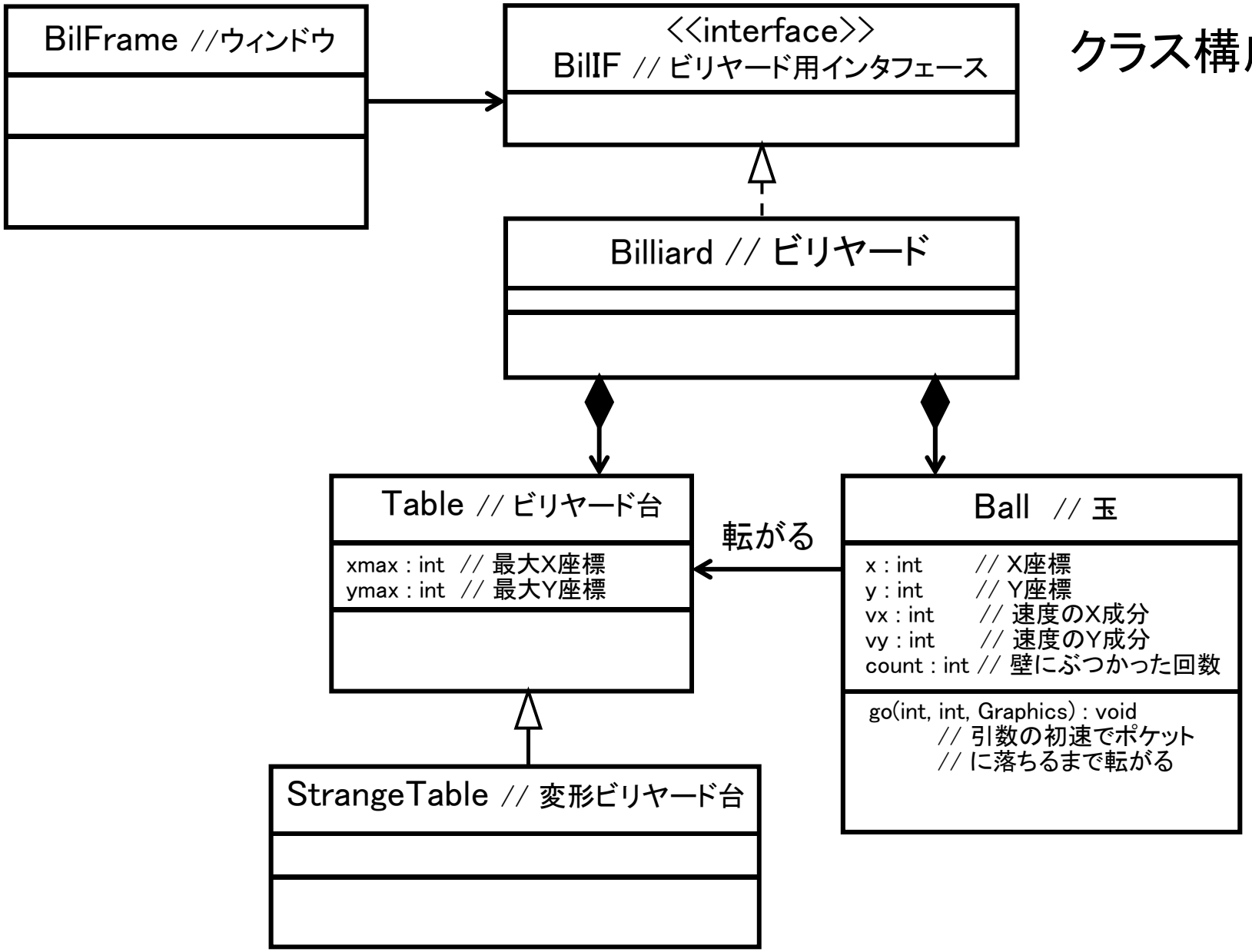
演習問題その2(2)

注意事項

- Billiard.java は、演習問題その2用のものをダウンロードして使用する。
  (ビリヤード台を選べるようになっている)
- クラス StrangeTable は、クラス Table を継承して作成する。
  (Tableで使える機能があれば、それを有効活用しよう)
- 通常のビリヤード台(Table)と変形ビリヤード台(StrangeTable)のいずれを選択しても正しく動くようにすること。

演習問題その2(3)

クラス構成



演習問題その2(4)

ビリヤード台の選択方法

(1) プログラムの起動法時に、ビリヤード台を選べるようになっている

```
>java Billiard
ビリヤード台の横方向の大きさは？ 800
ビリヤード台の縦方向の大きさは？ 390
テーブルの種類は(1:通常, 2:変形) 2
```

ビリヤード台の種類を指定する
1:通常のビリヤード台
2:変形ビリヤード台

(2) ビリヤード(Billiard.java)のコンストラクタで、指定されたビリヤード台のインスタンスを生成するようになっている

```
// コンストラクタ
// 第1引数:ビリヤード台の幅、第2引数:ビリヤード台の高さ
// 第3引数:ビリヤード台の種類(1のとき通常の台、2のとき変形台)
public Billiard(int m, int n, int t) {
  if (t == 1) {                          // 通常のビリヤード台の場合
    table = new Table (m, n);            // m×nサイズの通常のビリヤード台のインスタンスを生成
  } else {                               // 変形ビリヤード台の一場合
    table = new StrangeTable (m, n); // m×nサイズの変形ビリヤード台のインスタンスを生成
  }
  ball = new Ball(table, 0, 0);   // ボールのインスタンスを作成してテーブルの左上端に置く
}
```

プログラムの提出

「演習問題その２」のプログラムを完成できた場合は、Table.java、StrangeTable.java 、Ball.java の３つのソースファイル（「演習問題その１」の内容を含んでいる）を以下により圧縮し、ひとつのファイル src2.zip （注：ファイル名に'2'が入る）として提出してください。

```
zip src2.zip Table.java StrangeTable.java Ball.java
```

「演習問題その２」のプログラムを完成できなかった場合は、演習問題その１の Table.java、Ball.java の２つのソースファイルを以下により圧縮し、ひとつのファイル src1.zip （注：ファイル名に'1'が入る）として提出してください。

```
zip src1.zip Table.java Ball.java
```