プログラミング基礎

第8回
複数のクラスを使ったプログラム

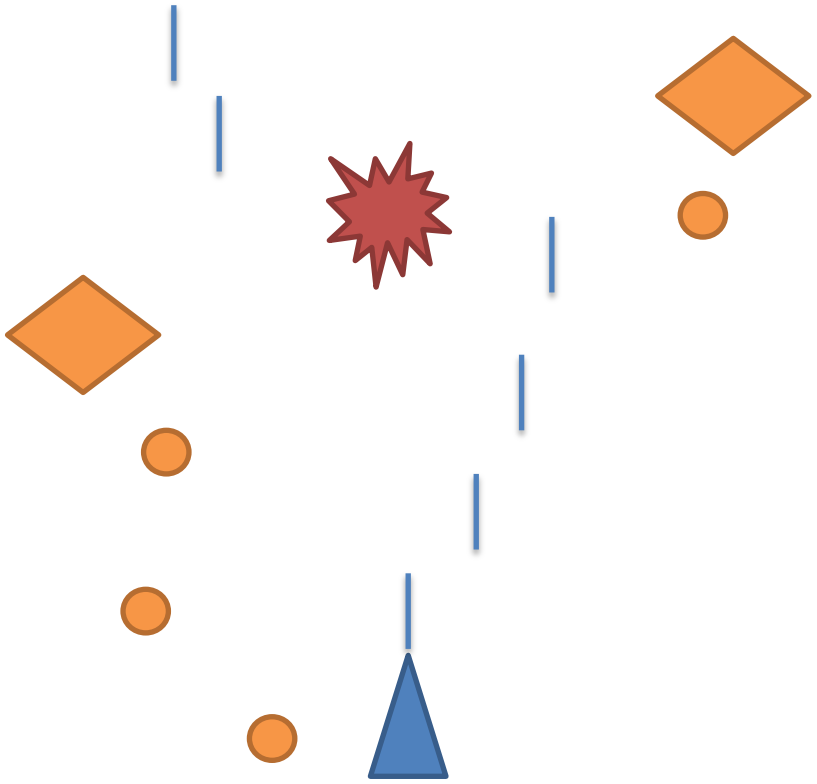
オブジェクトが別のオブジェクトを保有する: 包含（ほうがん）

- 要するに,
  「あるクラスAの属性の型にクラスBを用いる」
  ということ
  英語で”has-a”関係とも呼ばれる
  （クラスAはクラスBを持つ）

```
class クラスB
{
   int x;
}

class クラスA
{
   int foo;
   クラスB bar;
}
```

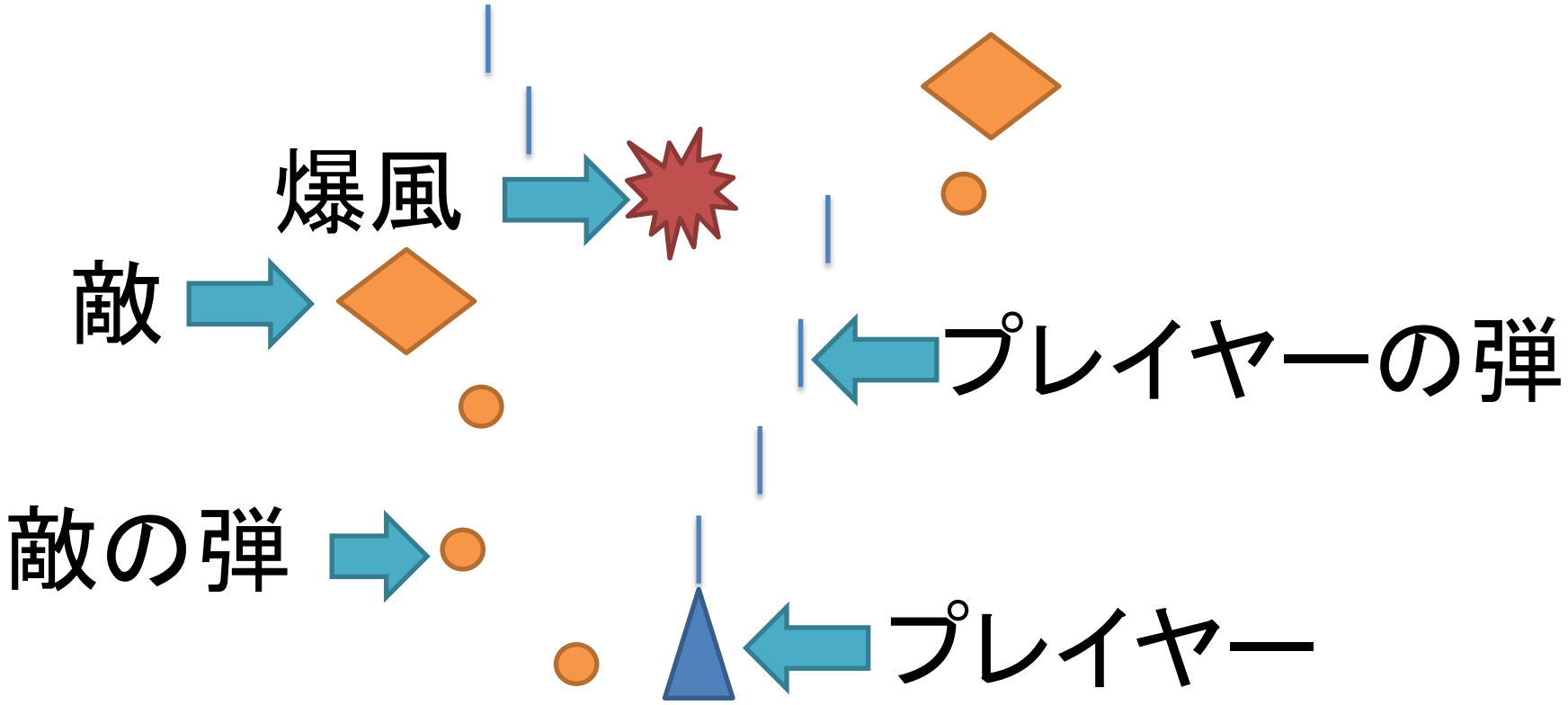
なぜクラスを使うのか

試しに以下のようなゲームを作ってみることを考える



なぜクラスを使うのか

試しに以下のようなゲームを作ってみることを考える



なぜクラスを使うのか

- ではクラスがなかったら…

```
static void main()
{
    int playerX;
    int playerY;
    int plyerLife;
    int enemyX = new int[10];
    int enemyY = new int[10];
    int enemyLife = new int[10];
    int playerBulletX = new int[10];
    int playerBulletY = new int[10];
    int playerBulletLife = new int[10];
    int enemyBulletX = new int[10];
    int enemyBulletY = new int[10];
    int enemyBulletLife = new int[10];
    int enemyBulletVx = new int[10];
    int enemyBulletVy = new int[10];
```

プレイヤーの位置

プレイヤーの体力

敵の位置

敵の体力

プレイヤーの弾の位置

プレイヤー弾の体力

敵の弾の位置

敵の弾が存在しているか

敵の弾の進行方向

スライドに入りきらないが，
他に，爆風用の位置などなど，同じようにすべて作っていくことになる

なぜクラスを使うのか

クラスを使うと…

```
static void main()
{
  int playerX;
  int playerY;
  int plyerLife;
  int enemyX = new int[10];
  int enemyY = new int[10];
  int enemyLife = new int[10];
  int playerBulletX = new int[10];
  int playerBulletY = new int[10];
  int playerBulletLife = new int[10];
  int enemyBulletX = new int[10];
  int enemyBulletY = new int[10];
  int enemyBulletLife = new int[10];
  int enemyBulletVx = new int[10];
  int enemyBulletVy = new int[10];
```

2次元での位置，向きを表す
⇒2次元ベクトルとしてまとめてしまえばよい

Vector2Dクラス（型）導入

```
class Vector2D
{
    int x;
    int y;
}
```

なぜクラスを使うのか

Vector2Dクラス（型）導入

```
static void main()
{
    Vector2D playerPosition;
    int plyerLife;
    Vector2D enemyPositions = new Vector2D[10];
    int enemyLife = new int[10];
    Vector2D playerBulletX = new Vector2D[10];
    int playerBulletLife = new int[10];
    Vector2D enemyBulletPositions = new Vector2D[10];
    int enemyBulletLife = new int[10];
    Vector2D enemyBulletVelocity = new Vector2D[10];
    Vector2D explosivePositions = new Vector2D[10];
    int explosiveLife = new int[10];
```

なぜクラスを使うのか

さらに見直せば,

```
static void main()
{
    Vector2D playerPosition;
    int plyerLife;
    Vector2D enemyPositions = new Vector2D[10];
    int enemyLife = new int[10];
    Vector2D playerBulletX = new Vector2D[10];
    int playerBulletLife = new int[10];
    Vector2D enemyBulletPositions = new Vector2D[10];
    int enemyBulletLife = new int[10];
    Vector2D enemyBulletVelocity = new Vector2D[10];
    Vector2D explosivePositions = new Vector2D[10];
    int explosiveLife = new int[10];
```

プレイヤーの情報

敵の情報

プレイヤーの弾の情報

敵の弾の情報

爆風の情報

さらにそれぞれをクラス化

なぜクラスを使うのか

```
static void main()
{
   Player player;
   Enemy enemies = new Enemy[10];
   PlayerBullet playerBullets = new PlayerBullets[10];
   EnemyBullets enemyBullets = new EnemyBullets[10];
   Explosive explosives = new Explosive[10];
```

```
class Player
{
   Vector2D position;
   int life;
}
```

```
class Enemy
{
   Vector2D position;
   Vector2D velocity;
   int life;
}
```

```
class Explosive
{
   Vector2D position;
   int life;
}
```

```
class PlayerBullet
{
   Vector2D position;
   int life;
}
```

```
class EnemyBullet
{
   Vector2D position;
   int life;
}
```

```
class Vector2D
{
   int x;
   int y;
}
```

どっちが分かりやすい？

```
static void main()
{
    int playerX;
    int playerY;
    int plyerLife;
    int enemyX = new int[10];
    int enemyY = new int[10];
    int enemyLife = new int[10];
    int playerBulletX = new int[10];
    int playerBulletY = new int[10];
    int playerBulletLife = new int[10];
    int enemyBulletX = new int[10];
    int enemyBulletY = new int[10];
    int enemyBulletLife = new int[10];
    int enemyBulletVx = new int[10];
    int enemyBulletVy = new int[10];
    int explosiveX = new int[10];
```

```
static void main()
{
    Player player;
    Enemy enemies = new Enemy[10];
    PlayerBullet playerBullets = new PlayerBullets[10];
    EnemyBullets enemyBullets = new EnemyBullets[10];
    Explosive explosives = new Explosive[10];
```

```
class Player
{
    Vector2D position;
    int life;
}

class Enemy
{
    Vector2D position;
    Vector2D velocity;
    int life;
}

class Explosive
{
    Vector2D position;
    int life;
}
```

```
class PlayerBullet
{
    Vector2D position;
    int life;
}

class EnemyBullet
{
    Vector2D position;
    int life;
}

class Vector2D
{
    int x;
    int y;
}
```

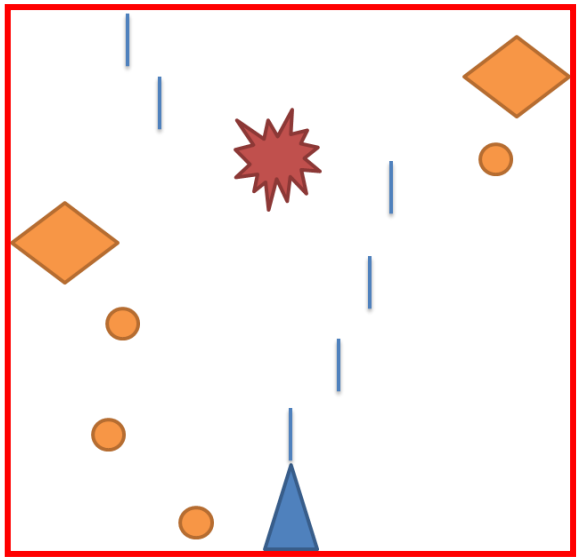
オブジェクト指向

- ひとつの大きなシステムを
  オブジェクト（部品）ごとに分けて考え，
  それらが組み合わさって動くものとする．
  プログラムも，それに沿って記述するという
  考え方

オブジェクト指向

Gameクラス

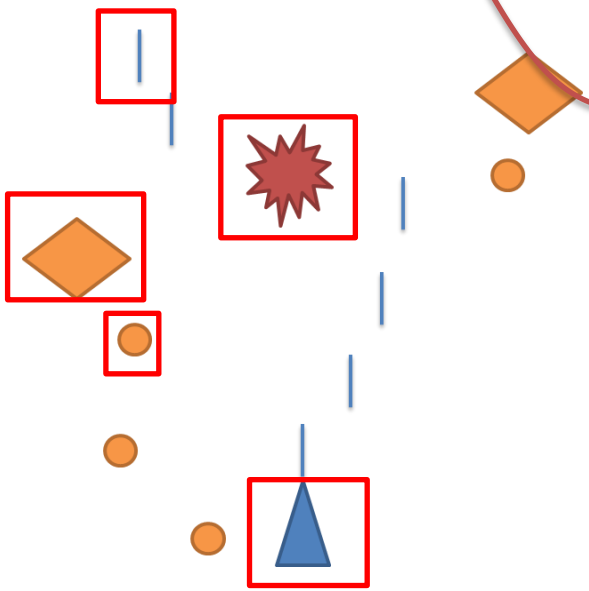
- ひとつの大きなシステムを
オブジェクト（部品）ごとに分けて考え，
それらが組み合わさって動くものとする
プログラムも，それに沿って記述するという
考え方



オブジェクト指向

- ひとつの大きなシステムを
  オブジェクト（部品）ごとに分けて考え，
  それらが組み合わさって動くものとする．
  プログラムも，それに沿って記述するという
  考え方

Gameクラス



GameクラスはPlayerクラス他を包含する

```
class Game
{
   Player player;
   Enemy [] enemies;
   …

   Game()
   {
      player = new Player();
      enemies = new Enemy[10];
      for(int i = 0; i < enemies.length; i++)
      {
          enemies[i] = new Enemy();
      }
      …
   }

   void Update()
   {
       player.Move();
       for(int i = 0; i < enemies.length; i++)
      {
           enemies[i].Move();
      }
   }
```

```
class Player
{
    Vector2D p;
    Player()
    {
    }

    void Move()
    {
        if(コントローラの左が押されてたら)
        {
            p.x -= 5.0;
        }
        else if (…
    }
}

class Enemy
{
    Vector2D p;
    Enemy()
    {
        …
```

```
public class GameMain
{
   static public void main(String[] args)
   {
      Game game = new Game();

      while(true)
      {
          game.Update();
      }
   }
}
```

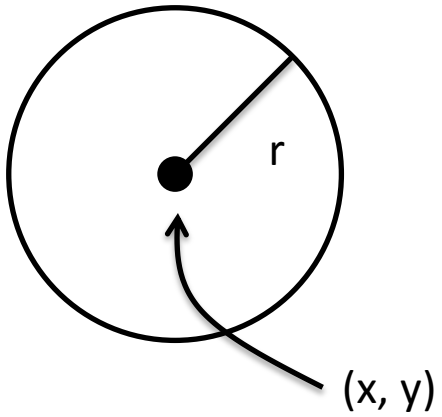
というわけで

- 包含という言葉よりも，
  「どのようにクラスを分ければ見やすい
  （分かりやすい）プログラムが書けるのか」
  慣れるのが今回の主題

包含の例

- Circleクラス（包含を用いない場合）

```
class Circle {
    double r; //半径
    double x; //円の位置x
    double y; //円の位置y

    //コンストラクタ
    Circle(double rr, double xx, double yy)
    {
        r = rr;
        x = xx;
        y = yy;
    }
}
```

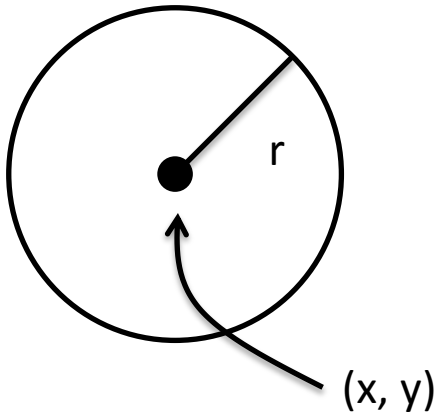


包含の例

- Circleクラス（包含を用いない場合）

```
class Circle {
    double r; //半径
    double x; //円の位置x
    double y; //円の位置y

    //コンストラクタ
    Circle(double rr, double xx, double yy)
    {
        r = rr;
        x = xx;
        y = yy;
    }
}
```

位置は2次元ベクトルで表せばよい

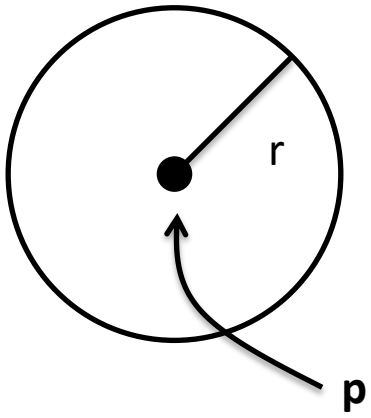


包含の例

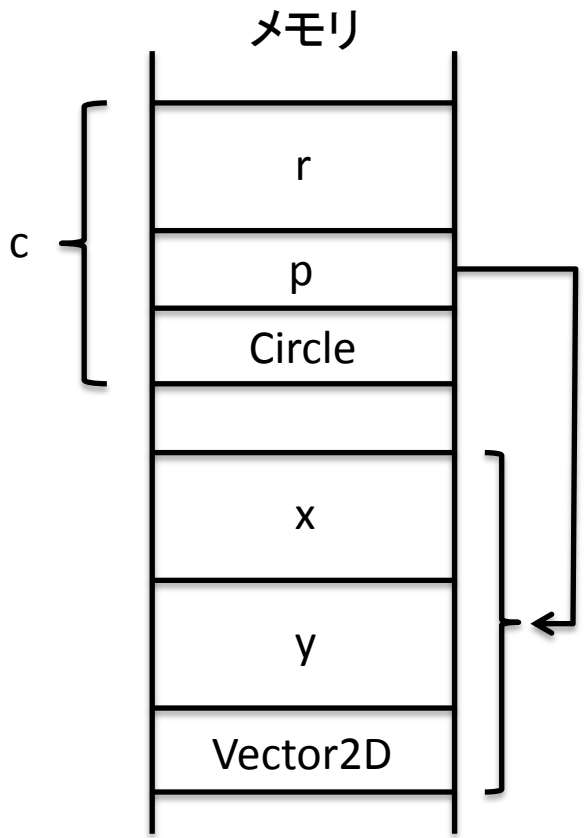
- Circleクラス（包含を用いた場合）

```
class Vector2D
{
   double x;
   double y;

   //コンストラクタ
   Vector2D(double xx, double yy)
   {
      x = xx;
      y = yy;
   }
}
```

```
class Circle {
   double r; //半径
   Vector2D p; //円の位置p
   //コンストラクタ
   Circle(double rr, double xx, double yy)
   {
      r = rr;
      p = new Vector2D(xx, yy);
   }
}
```



```
public class CircleTest
{
   static public void main(String[] args)
   {
      Circle c = new Circle(5.0, 0.0, 0.0);
      //半径5.0の円を原点に作成
      //（インスタンスを作成）
   }
}
```



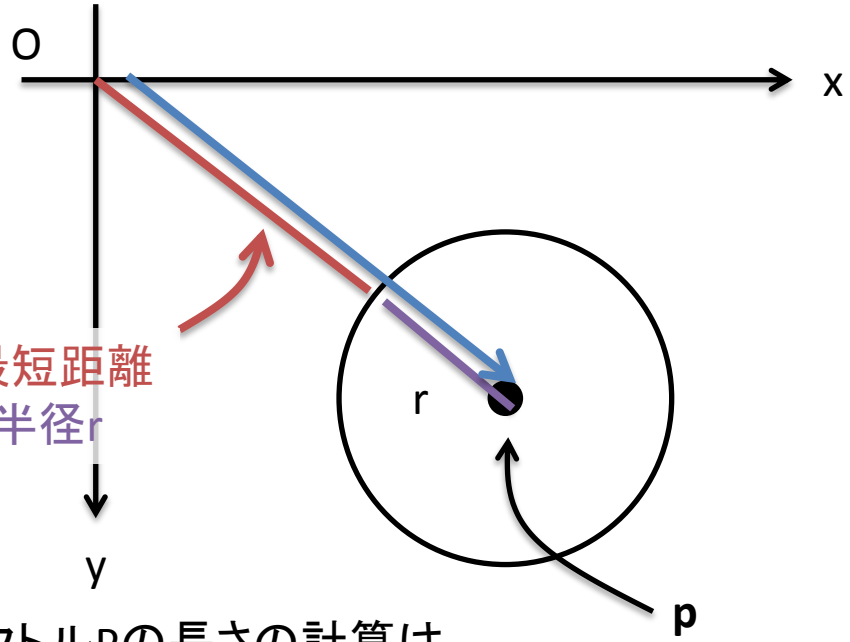

```
class Vector2D
{
   double x;
   double y;

   //コンストラクタ
   Vector2D(double xx, double yy)
   {
      x = xx;
      y = yy;
   }
}

class Circle {
   double r; //半径
   Vector2D p; //円の位置p
   //コンストラクタ
   Circle(double rr, double xx, double yy)
   {
      r = rr;
      p = new Vector2D(xx, yy);
   }
}
```

演習

- 原点から円までの最短距離を戻り値とするgetLengthFromOriginメソッドをCircleクラスに追加せよ（プログラム名は前スライドの例：CircleTestのままでよい）

- 書くコードの量が多くなるのでget～メソッドは作成しなくても良い

演習解き方



原点から円までの最短距離
=ベクトルPの長さ－半径r

ベクトルPの長さの計算は，
Circleクラスではなく，Vector2Dの仕事とする
⇒Vector2Dに自身の長さを戻り値とするgetLengthメソッドを追加する

平方根の計算はMath.sqrt
double x = Math.sqrt(9); //xは3

演習解き方

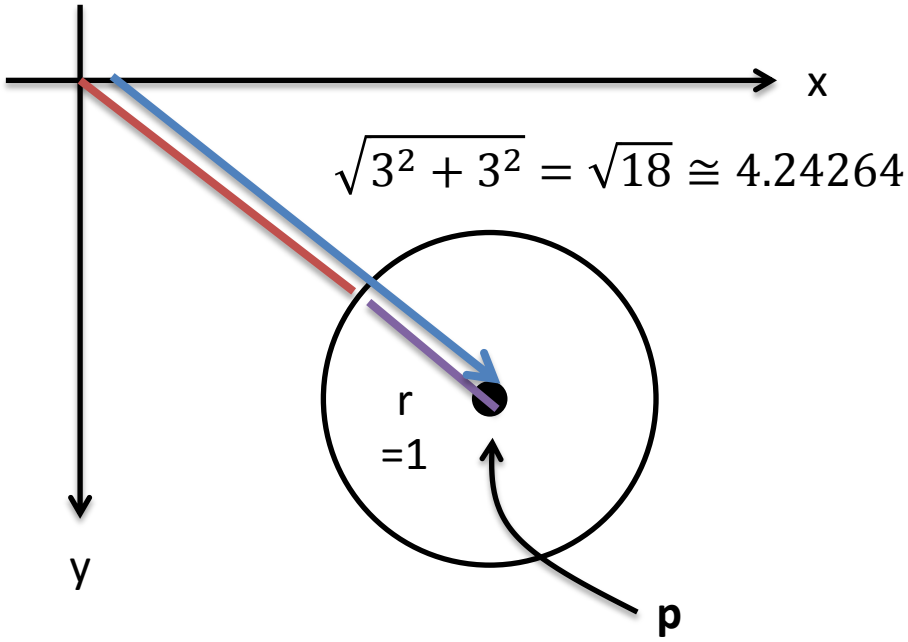
各クラスに適切なメソッドがあれば，mainメソッドは以下でOK

```
public class CircleTest
{
   static public void main(String[] args)
   {
      Circle c = new Circle(1.0, 3.0, 3.0);
      double r = c.r;
      double x = c.p.x;
      double y = c.p.y;
      System.out.println("球の半径:" + r + " 位置x:" + x + " 位置y:" + y);

      double l = c.getLengthFromOrigin();
      System.out.println("原点から球までの最短距離:" + l);
   }
}
```

O

演習の解き方

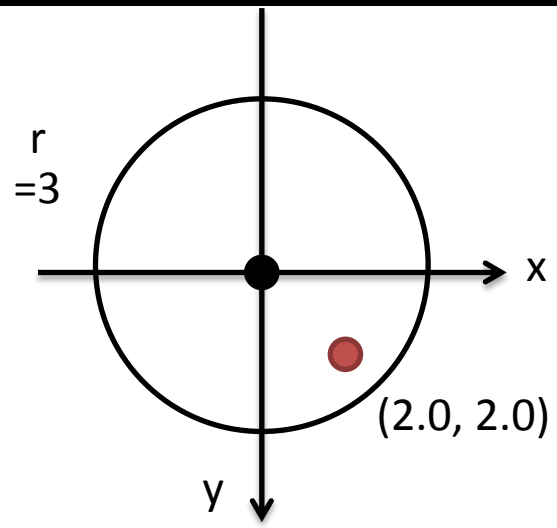
```
C:¥Users¥moriya¥Desktop>java CircleTest
球の半径：1.0 位置x：3.0 位置y：3.0
原点から球までの最短距離：3.2426406871192848
```



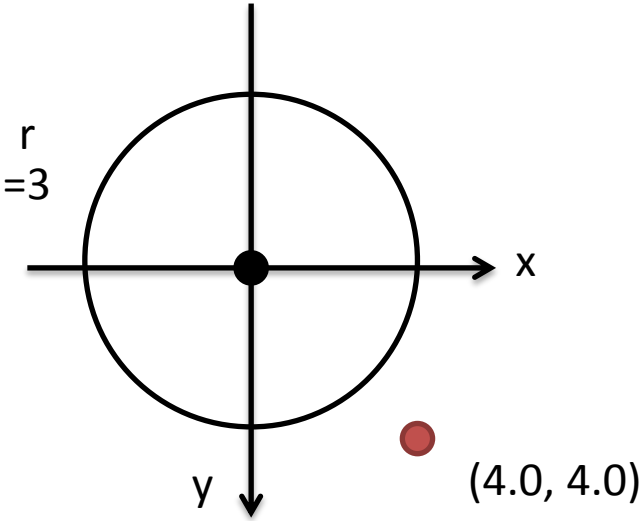
時間が余った人向け演習問題

- 引数に指定された点が，円の中にあるか，boolean型で返すisInCircleメソッドを作成せよ
  （true：引数の点は円の中，false：円の外）

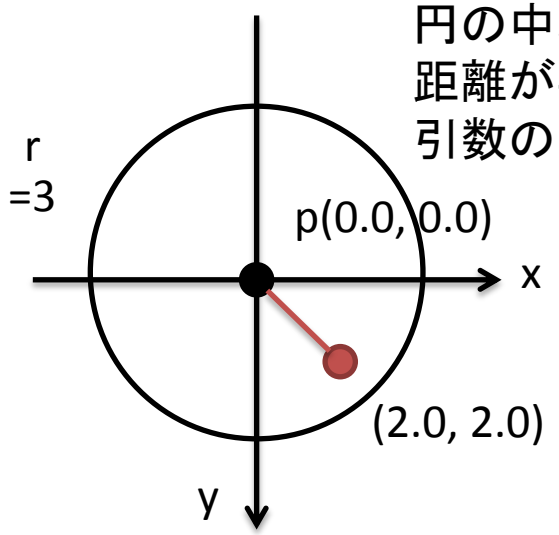
```
C:¥Users¥moriya¥Desktop>java CircleTest
円の半径：3.0 位置x：0.0 位置y：0.0
点(2.0, 2.0)は円の中です
```




```
C:¥Users¥moriya¥Desktop>java CircleTest
円の半径：3.0 位置x：0.0 位置y：0.0
点(4.0, 4.0)は円の外です
```



時間が余った人向け演習問題



円の中心pと引数に指定された点間の
距離が半径r以下であれば
引数の点は円の中

⇒Vector2Dに自身と引数に指定された点間の距離を戻り値とするgetLengthメソッドを作る（オーバーロード）

```
public class CircleTest
{
   static public void main(String[] args)
   {
      Circle c = new Circle(3.0, 0.0, 0.0);
      double r = c.r;
      double x = c.p.x;
      double y = c.p.y;
      System.out.println("円の半径:" + r + " 位置x:" + x + " 位置y:" + y);

      double qx = 4.0;
      double qy = 4.0;

      System.out.print("点(" + qx + ", " + qy + ")は円の");
      if (c.isInCircle(qx, qy))
      {
         System.out.println("中です");
      }
      else
      {
         System.out.println("外です");
      }
   }
}
```