



プログラミング演習2

継承

中村, 小松, 小林, 橋本



- 猫, 犬, 猿, 象, 熊を定義

- それぞれの座標は意識したくない

- `cat.x`, `cat.y`, `dog.x`, `dog.y`, `monkey.x`, `monkey.y`, ...
- 内部で適当に処理してもらう

- 描画はシンプルにしたい

- `cat.draw()`, `dog.draw()`, `monkey.draw()`, `elephant.draw()`, ...

- 移動もシンプルにしたい

- `cat.move()`, `dog.move()`, `monkey.move()`, `elephant.move()`, ...

- 睡眠も任せてしまう

- `cat.sleep()`, `dog.sleep()`, `monkey.sleep()`, `elephant.sleep()`, ...

いいんだけど、重複があるのでは？

先々週の宿題



- Ball クラスを改良し，☆が動き回るStarクラスを作成せよ．また，BallクラスとStarクラスを利用して，50個の丸と50個の星を動かすようにせよ．
 - 星の内部は塗りつぶせるようだったら塗りつぶせ

sketch_140928b

ほとんど重複している！
違うのはdisplay()だけ！
もっと簡単にできないの？

!!! 継承 !!!



• 大辞林 第三版

1. 先の人 の 身分・権利・義務・財産などを受け継ぐこと。「王位を一する」
2. インヘリタンス → (オブジェクト指向プログラミングにおいて、クラス間でデータの共有を行う機構。新しく定義するクラスを既存のクラスの下位クラスとして記述し、上位クラスより属性やメソッドを引き継ぐ仕組みをいう。上位クラスに対する差分のみを記述するだけで新しいクラスを定義することが可能となる。)

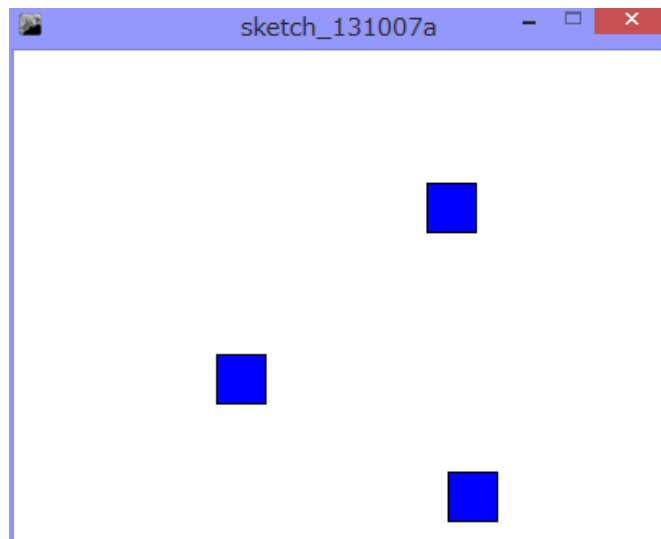
端で跳ね返る正方形を描く



(Q1) 400x300のウィンドウ内で、任意の場所から毎フレームx方向, y方向に任意の速度で移動する3つの青色の四角形を描画し、右端・左端・上端・下端に來ると跳ね返るようにするには？

• 考え方

- 右端・左端・上端・下端で衝突する時の条件を整理
- 衝突した時の速度を反転させる
 - $speedX = -speedX;$
 - $speedY = -speedY;$



Ballを改良しSquareを作る

明治大学総合数理学部
先端メディアサイエンス学科
中村研究室



```
class Square{
  int x;
  int y;
  int speedX;
  int speedY;

  Square(){
    init();
  }
  void init(){
    x = (int)random(width);
    y = (int)random(height);
    speedX = (int)random(5);
    speedY = (int)random(5);
  }
  void display(){
    fill( 0, 0, 255 );
    rect( x-15, y-15, 30, 30 );
  }
}
```

```
void move(){
  x = x + speedX;
  y = y + speedY;
  if ( x+15 > width ) {
    x = width - 15;
    speedX = -speedX;
  }
  if( x - 15 < 0 ){
    x = 15;
    speedX = -speedX;
  }
  if( y + 15 > height ){
    y = height - 15;
    speedY = -speedY;
  }
  if( y - 15 < 0 ){
    y = 15;
    speedY = -speedY;
  }
}
}
```

Squareクラスを使うと



```
Square fukuchi;
Square nakamura;
Square hashimoto;
void setup() {
    size( 400, 300 );
    fukuchi = new Square();
    nakamura = new Square();
    hashimoto = new Square();
}

void draw() {
    background(255);
    fukuchi.move();
    nakamura.move();
    hashimoto.move();
    fukuchi.display();
    nakamura.display();
    hashimoto.display();
}
```

3つの四角形と3つの丸



(Q2) 400x300のウィンドウ内で, 任意の場所から
毎フレームx方向, y方向に任意の速度で移動す
る赤色の3つの丸と, 青色の3つの四角形を描画
し, 右端・左端・上端・下端に來ると跳ね返るよう
にするには?

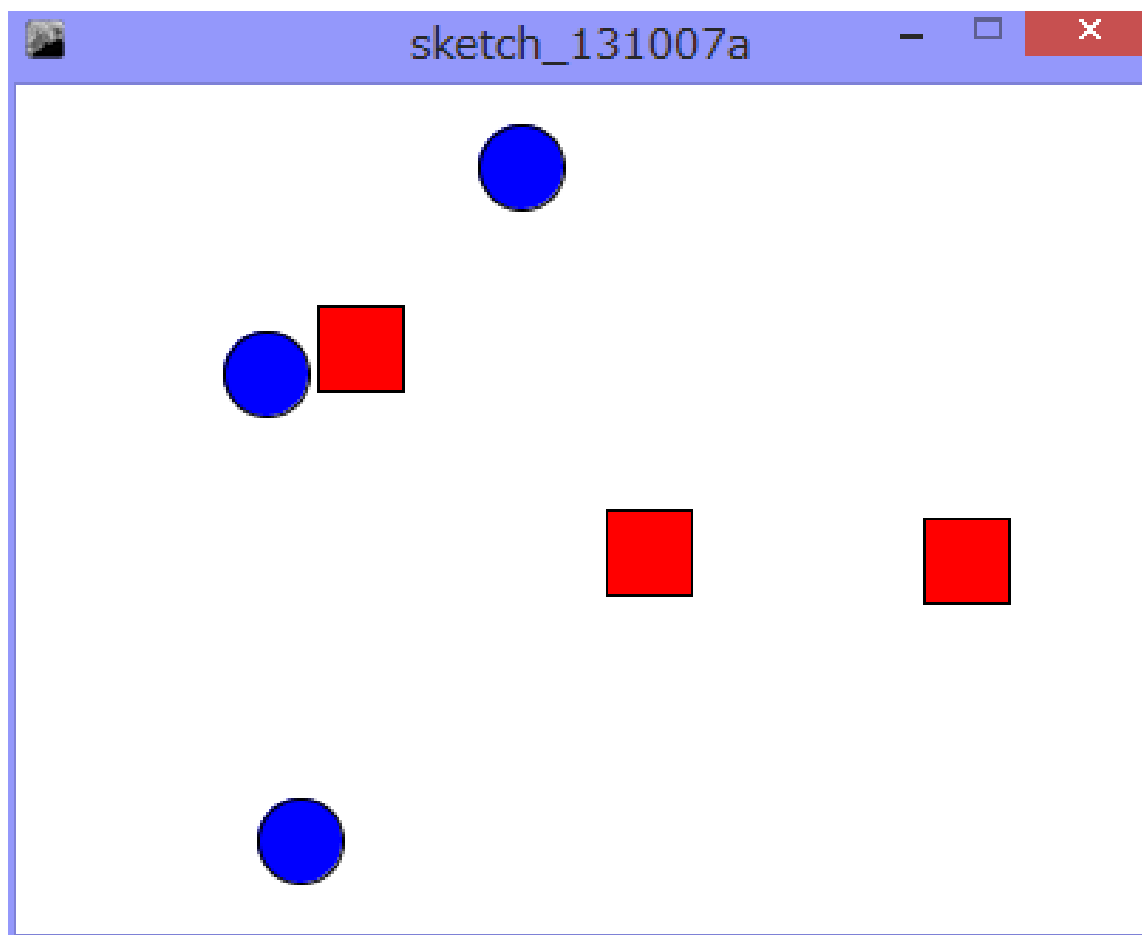
• 考え方

- 右端・左端・上端・下端で衝突する時の条件を整理
- 衝突した時の速度を反転させる
 - $speedX = -speedX;$
 - $speedY = -speedY;$

3つの四角形と3つの丸



- 前に作ったBallクラスと、今回作ったSquareクラスを組み合わせる



Ballクラス



```
class Ball{
  int x;
  int y;
  int speedX;
  int speedY;

  Ball(){
    init();
  }
  void init(){
    x = (int)random(width);
    y = (int)random(height);
    speedX = (int)random(5);
    speedY = (int)random(5);
  }
  void display(){
    fill( 255, 0, 0 );
    ellipse( x, y, 30, 30 );
  }
}
```

```
void move(){
  x = x + speedX;
  y = y + speedY;
  if ( x+15 > width ) {
    x = width - 15;
    speedX = -speedX;
  }
  if( x - 15 < 0 ){
    x = 15;
    speedX = -speedX;
  }
  if( y + 15 > height ){
    y = height - 15;
    speedY = -speedY;
  }
  if( y - 15 < 0 ){
    y = 15;
    speedY = -speedY;
  }
}
```

Ballを改良しSquareを作る



```
class Square{
  int x;
  int y;
  int speedX;
  int speedY;

  Square(){
    init();
  }
  void init(){
    x = (int)random(width);
    y = (int)random(height);
    speedX = (int)random(5);
    speedY = (int)random(5);
  }
  void display(){
    fill( 0, 0, 255 );
    rect( x-15, y-15, 30, 30 );
  }
}
```

```
void move(){
  x = x + speedX;
  y = y + speedY;
  if ( x+15 > width ) {
    x = width - 15;
    speedX = -speedX;
  }
  if( x - 15 < 0 ){
    x = 15;
    speedX = -speedX;
  }
  if( y + 15 > height ){
    y = height - 15;
    speedY = -speedY;
  }
  if( y - 15 < 0 ){
    y = 15;
    speedY = -speedY;
  }
}
}
```

3つの円と3つの四角形を移動

明治大学総合数理学部
先端メディアサイエンス学科
中村研究室



```
Ball miyashita;
Ball komatsu;
Ball kikuchi;
Square fukuchi;
Square nakamura;
Square hashimoto;
void setup() {
    size( 400, 300 );
    fill( 255, 0, 0 );
    miyashita = new Ball();
    komatsu = new Ball();
    kikuchi = new Ball();
    fukuchi = new Square();
    nakamura = new Square();
    hashimoto = new Square();
}
```

```
void draw() {
    background(255);
    // 移動
    miyashita.move();
    komatsu.move();
    kikuchi.move();
    fukuchi.move();
    nakamura.move();
    hashimoto.move();
    // 描画
    miyashita.display();
    komatsu.display();
    kikuchi.display();
    fukuchi.display();
    nakamura.display();
    hashimoto.display();
}
```

ここで...



- Ballクラスと, Squareクラスはほとんど一緒
- 違いはクラス名とコンストラクタ, そしてdisplayのインスタンスメソッドのみ

無駄じゃね？

ほとんど一緒



	Ball	Square	
インスタンス変数	x, y speedX, speedY	x, y speedX, speedY	一致
クラス名	Ball	Square	違う
コンストラクタ	Ball()	Square()	内部は一致
void init()			一致
void move()			一致
void display()	fill(0, 0, 255); ellipse(x, y, 30, 30);	fill(0, 0, 255); rect(x-15, y-15, 30, 30);	違う

一緒の部分をまとめた
スーパークラス(親クラス)を作る

スーパーなObjectクラス



```
class Object{
  int x;
  int y;
  int speedX;
  int speedY;

  Object(){
    init();
  }
  void init(){
    x = (int)random(width);
    y = (int)random(height);
    speedX = (int)random(5);
    speedY = (int)random(5);
  }
}
```

display() は内容が違うので
削除してしまう

```
void move(){
  x = x + speedX;
  y = y + speedY;
  if ( x+15 > width ) {
    x = width - 15;
    speedX = -speedX;
  }
  if( x - 15 < 0 ){
    x = 15;
    speedX = -speedX;
  }
  if( y + 15 > height ){
    y = height - 15;
    speedY = -speedY;
  }
  if( y - 15 < 0 ){
    y = 15;
    speedY = -speedY;
  }
}
```

一緒に部分をまとめる



- Ball は Object の変数 (x , y , $speedX$, $speedY$) や機能 (移動や初期化) をもち, 独自の表示に関する機能 (メソッド) をもつクラス
- Square は Object の変数 (x , y , $speedX$, $speedY$) や機能 (移動や初期化) をもち, 独自の表示に関する機能 (メソッド) をもつクラス
- インスタンス変数や, インスタンスメソッドを引き継ぐことを**継承**と呼ぶ!

Objectクラスを使うと



```
class Ball extends Object {  
    void display(){  
        fill( 255, 0, 0 );  
        ellipse( x, y, 30, 30 );  
    }  
}  
  
class Square extends Object {  
    void display(){  
        fill( 0, 0, 255 );  
        rect( x-15, y-15, 30, 30 );  
    }  
}
```



BallクラスとSquareクラスが劇的に短く！

```
Ball miyashita;  
Ball komatsu;  
Ball kikuchi;  
Square fukuchi;  
Square nakamura;  
Square hashimoto;  
void setup() {  
    size( 400, 300 );  
    fill( 255, 0, 0 );  
    miyashita = new Ball();  
    komatsu = new Ball();  
    kikuchi = new Ball();  
    fukuchi = new Square();  
    nakamura = new Square();  
    hashimoto = new Square();  
}
```

なぜ動くの？



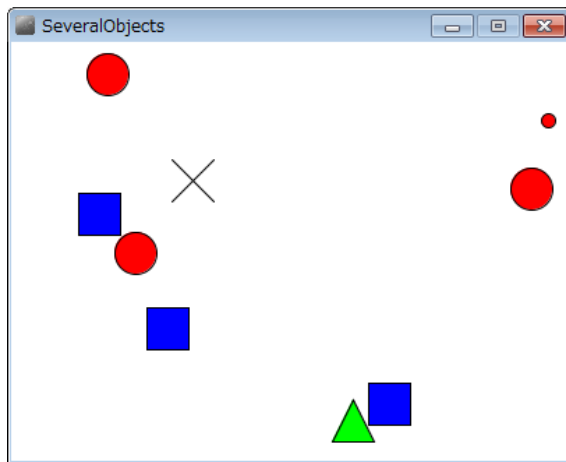
- 継承すると、親の力をすべて引き継ぐ！
- 継承の方法は extends とやるだけ！

```
class クラス名 extends 親クラス名 {  
  
}
```

- 継承により親の能力，値はすべて引き継ぎます



- Object クラスを継承して「三角形」を描画するクラスを作るには？ (Triangle)
- Object クラスを継承して「×」を描画するクラスを作るには？ (Cross)



Object

x, y
speedX
speedY

init()

move()

Object()

Triangle

display()

三角形と×のクラス



- 動く三角形のクラスと動く×のクラス

```
class Triangle extends Object {  
    void display(){  
        fill( 0, 255, 0 );  
        triangle( x, y-15, x-15, y+15, x+15, y+15 );  
    }  
}
```

```
class Cross extends Object {  
    void display(){  
        fill( 0, 255, 0 );  
        line( x-15, y-15, x+15, y+15 );  
        line( x-15, y+15, x+15, y-15 );  
    }  
}
```

つまり先々週の宿題は...



- Ball クラスを改良し，☆が動き回るStarクラスを作成せよ. また，BallクラスとStarクラスを利用して，50個の丸と50個の星を動かすようにせよ.
 - 星の内部は塗りつぶせるようだったら塗りつぶせ



Objectクラスを継承して
Starクラスを作り，displayだけ
独立させる！



三角形は跳ね返らないように



(Q3) 先述のObjectを継承したTriangleクラスを改良し, 三角形は跳ね返らず右端→左端, 左端→右端, 上端→下端, 下端→上端と移動するようにせよ

• 考え方

- Object の move メソッドを Triangle クラス内でオーバーライドして, 三角形専用のメソッドを作成する

```
class Triangle extends Object {  
    void display(){  
        fill( 0, 255, 0 );  
        triangle( x, y-15, x-15, y+15, x+15, y+15 );  
    }  
    void move(){  
        x = x + speedX;  
        y = y + speedY;  
        if( x > width ){  
            x = x - width;  
        } else if( x < 0 ){  
            x = width + x;  
        }  
        if( y > height ){  
            y = y - height;  
        }  
        if( x < 0 ){  
            y = width + y;  
        }  
    }  
}
```

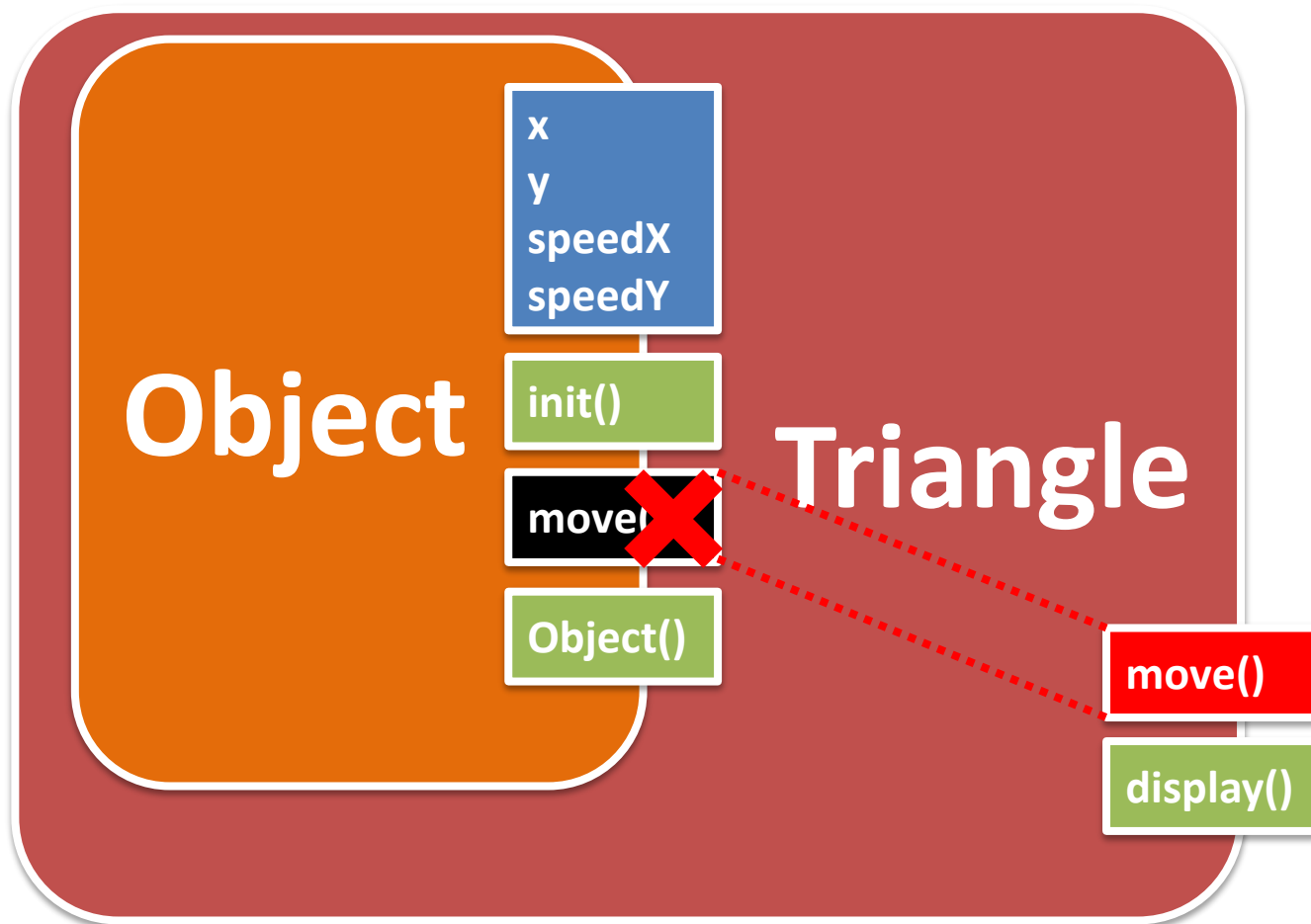


move メソッドをオーバーライドして
親の move メソッドが呼ばれないよ
うにする

move をオーバーライド



- Triangle の move で, Object の move を上書きしてしまう!



全体の挙動を変更



(Q4) Objectクラスのmoveメソッドを変更し，上端及び下端は跳ね返るが，左端と右端では逆側から現れるようにせよ

• 考え方

- Object の move メソッドのみ変更
- y 座標の条件で跳ね返り
- x 座標の条件で逆側から現れるようにする

Object クラスのみ変更



```
class Object{
  int x;
  int y;
  int speedX;
  int speedY;

  Object(){
    init();
  }
  void init(){
    x = (int)random(width);
    y = (int)random(height);
    speedX = (int)random(5);
    speedY = (int)random(5);
  }
}
```

```
void move(){
  x = x + speedX;
  y = y + speedY;
  if( x > width ){
    x = x - width;
  } else if( x < 0 ){
    x = width + x;
  }

  if( y + 15 > height ){
    y = height - 15;
    speedY = -speedY;
  } else if( y - 15 < 0 ){
    y = 15;
    speedY = -speedY;
  }
}
```

惑星と衛星の様なオブジェクト



(Q5) 400x300のウィンドウ内で, 任意の場所 x, y から任意の速度で移動する3つの赤色の円を描画し, 右端・左端・上端・下端に来ると跳ね返るようになる. また, 赤色の円には円の中心から30の距離があるところに1つの衛星があり, 10度ずつ円の周りを回転するようにせよ

• 考え方

– 円の中心 (x, y) から衛星の方向の角度 $(0 \sim 360$ 度)を θ とすると, 衛星の座標は

$(x+30 \cdot \cos(\text{radians}(\theta)), y+30 \cdot \sin(\text{radians}(\theta)))$

惑星と衛星の様なオブジェクト



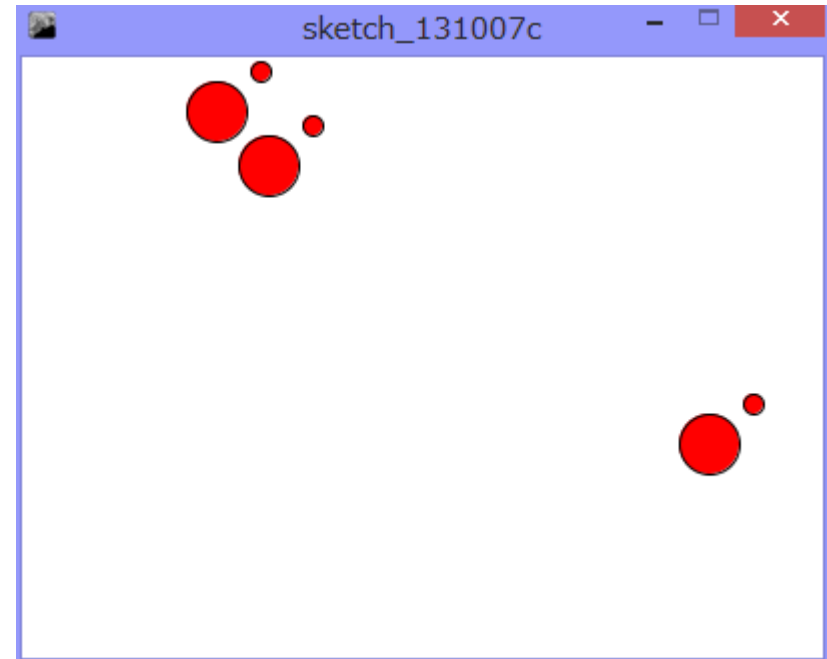
- Objectクラスを継承して, 変数を追加する

```
class PlanetSatellite extends Object {  
    int theta;  
    void display(){  
        fill( 255, 0, 0 );  
        ellipse( x, y, 30, 30 );  
        theta = theta + 10;  
        int rx = (int)(x+30*sin(radians(theta)));  
        int ry = (int)(y+30*cos(radians(theta)));  
        ellipse( rx, ry, 10, 10 );  
    }  
}
```

惑星と衛星の様なオブジェクト



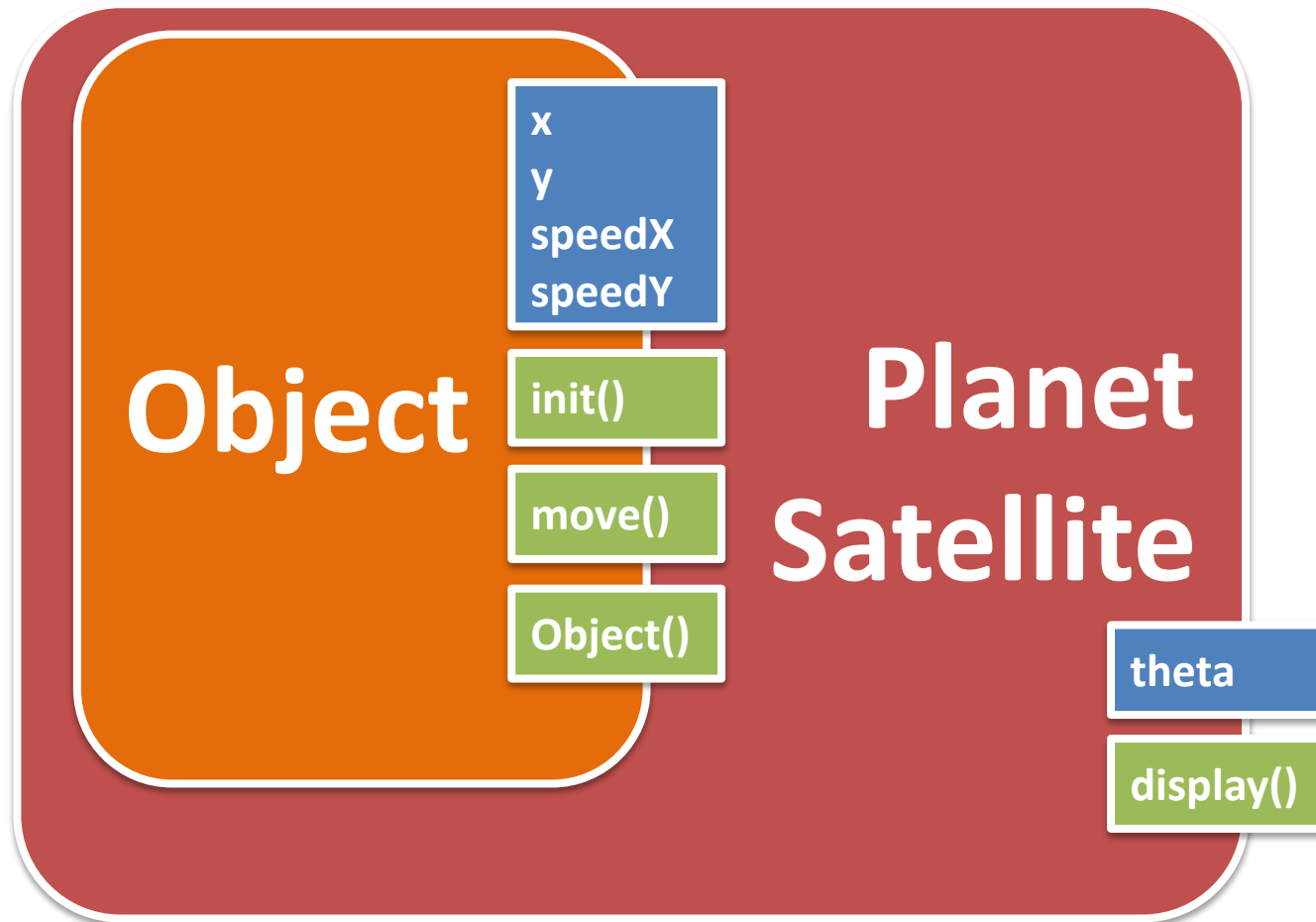
```
PlanetSatellite miyashita;  
PlanetSatellite komatsu;  
PlanetSatellite kikuchi;  
void setup() {  
  size( 400, 300 );  
  miyashita = new PlanetSatellite();  
  komatsu = new PlanetSatellite();  
  kikuchi = new PlanetSatellite();  
}  
  
void draw() {  
  background(255);  
  miyashita.move();  
  komatsu.move();  
  kikuchi.move();  
  miyashita.display();  
  komatsu.display();  
  kikuchi.display();  
}
```



継承すると...



- Object を PlanetSatellite として継承し, theta という変数と, display() というメソッドを追加



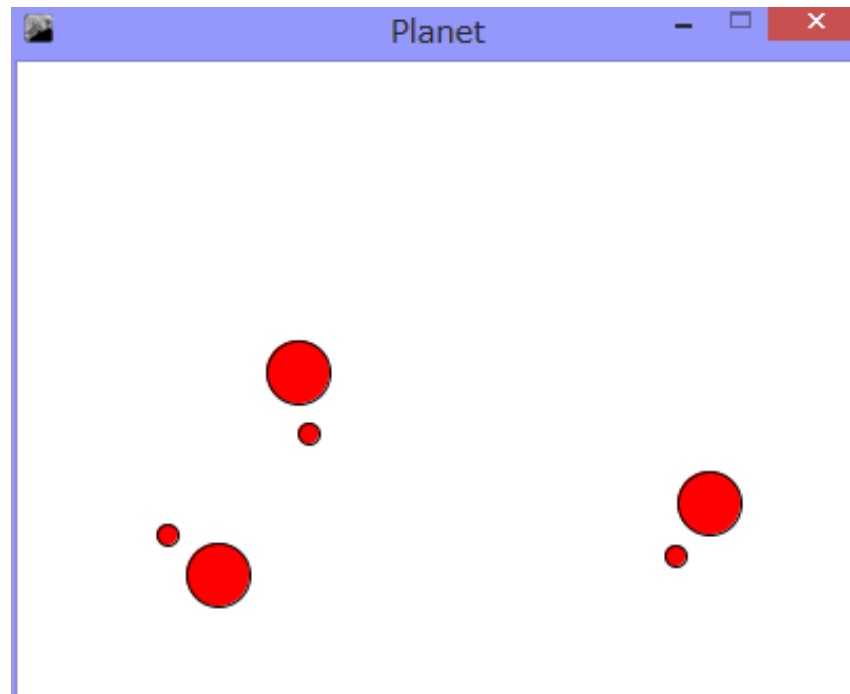
惑星と衛星の様なオブジェクト



(Q5) Q4を改良し, 衛星の開始角を0~360度の任意の場所にしたい. どうするか?

- 考え方

- void init() というインスタンスメソッドを追加し, 改良したらOK?



やってみる



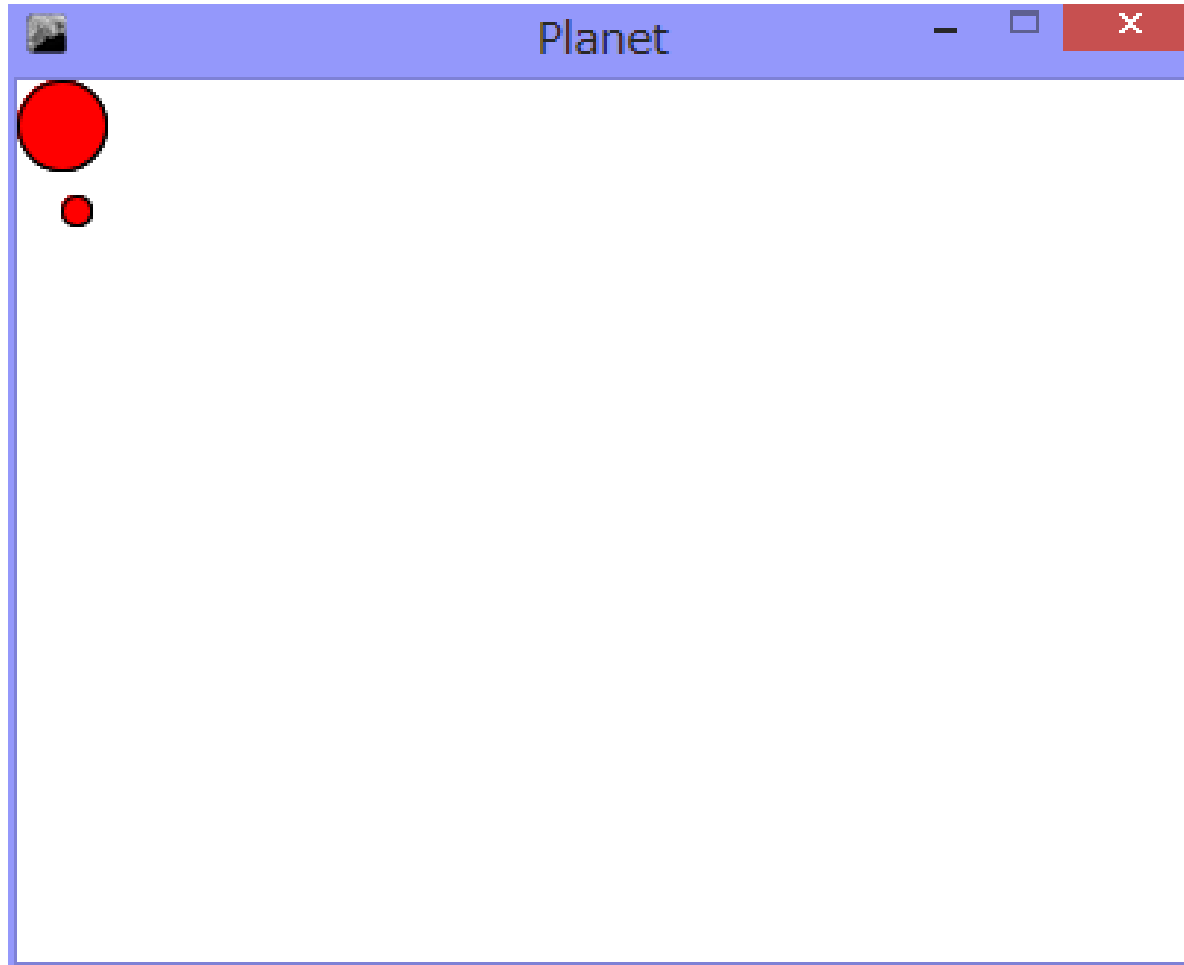
```
class PlanetSatellite extends Object {  
    int theta;  
    void init(){  
        theta = (int)random(360);  
    }  
    void display(){  
        fill( 255, 0, 0 );  
        ellipse( x, y, 30, 30 );  
        theta = theta + 10;  
        int rx = (int)(x+30*sin(radians(theta)));  
        int ry = (int)(y+30*cos(radians(theta)));  
        ellipse( rx, ry, 10, 10 );  
    }  
}
```

init() はコンストラクタで呼ばれる
Objectクラス参照

うまく動作しない



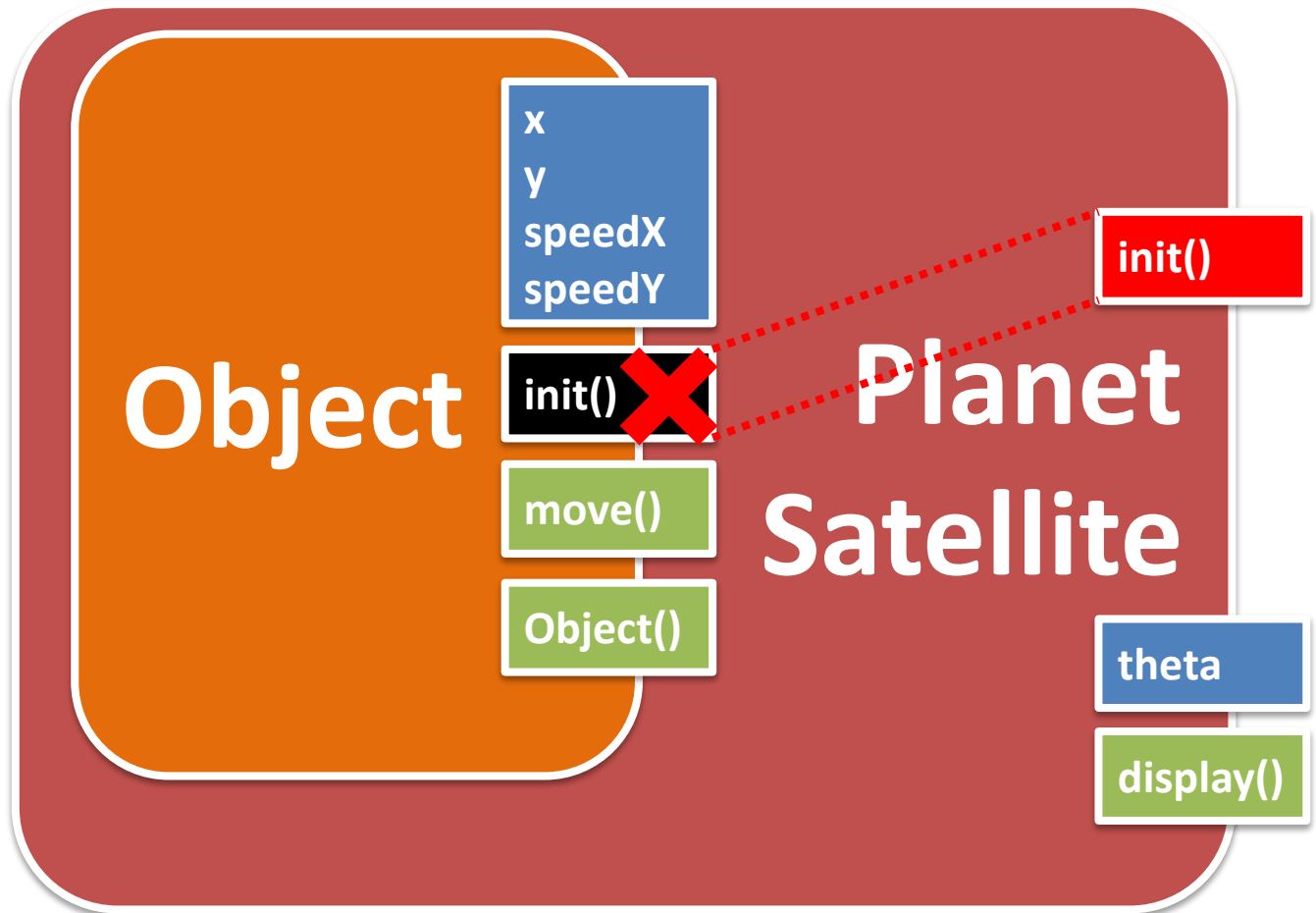
- 画面の左上から動かない. 何故か？



上書きしてしまったから



- Object の `init()` を, PlanetSatellite の `init()` でオーバーライドしている (Objectの`init()`が呼び出されない)





```
class PlanetSatellite extends Object {
```

```
    int theta;
```

```
    void init(){
```

```
        theta = (int)random(360);
```

```
        x = (int)random(width);
```

```
        y = (int)random(height);
```

```
        speedX = (int)random(5);
```

```
        speedY = (int)random(5);
```

```
    }
```

```
    void display(){
```

```
        fill( 255, 0, 0 );
```

```
        ellipse( x, y, 30, 30 );
```

```
        theta = theta + 10;
```

```
        int rx = (int)(x+30*sin(radians(theta)));
```

```
        int ry = (int)(y+30*cos(radians(theta)));
```

```
        ellipse( rx, ry, 10, 10 );
```

```
    }
```

```
}
```

Object の init() にあるのを
そのままコピーする

動くけど、なんだか
無駄が増えている...

Objectのinit()も使いたい



- クラスの中で, super と書くと, 継承元の親を呼び出すことができる!

```
class PlanetSatellite extends Object {  
    int theta;  
    void init(){  
        theta = (int)random(360);  
        super.init();  
    }  
    void display(){  
        fill( 255, 0, 0 );  
        ellipse( x, y, 30, 30 );  
        theta = theta + 10;  
        int rx = (int)(x+30*sin(radians(theta)));  
        int ry = (int)(y+30*cos(radians(theta)));  
        ellipse( rx, ry, 10, 10 );  
    }  
}
```

super . メソッド名

課題4-1 衛星を3つにする

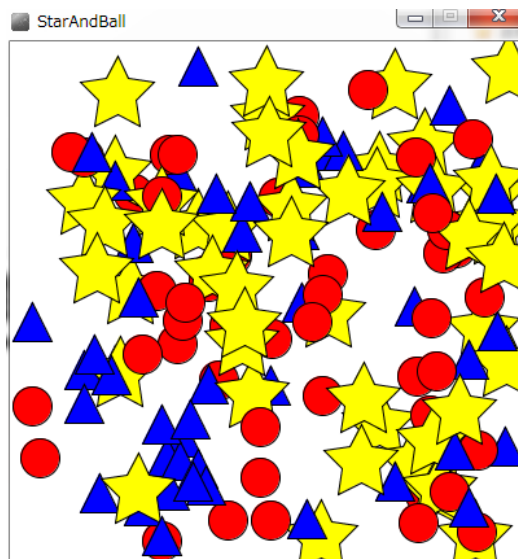


- Q4の PlanetSatellite クラスを継承し，衛星の数を3つにする PlanetSatellite3 クラスを作成せよ．また，そのクラスを用いて10個の惑星が動きまわるようにせよ
 - 考え方
 - インスタンス変数を追加する
 - init メソッドと， display メソッドをオーバーライドする

課題4-2 BallStarTriangle



- Objectクラスを継承し，☆が動き回るStarクラスを作成せよ．また，BallクラスとStarクラス，Triangleクラスを利用して，50個の赤色丸と50個の黄色星と50個の青色△を動かすようにせよ
 - 星の内部は塗りつぶせないようだったら，線の色を黄色にせよ



宿題1



- 先々週作成した掲示板クラスを継承し，丸形を星形にした掲示板クラス KeijibanEx を作成せよ
- また，クリック回数に応じて色が白→青→黄→赤(→白)と変化するようにせよ
- さらに，scrollUp(下から上へ)とscrollDown(上から下へ)というメソッドも追加せよ



- 100秒タイマークラス (TimerBase) を利用して残り時間を表示してみよう
 - ただし、タイマーはクリックの度にカウントダウンを開始し、0秒になったら停止するようにせよ
 - TimerBaseは講義資料サイトからダウンロードしよう
 - また、100秒タイマークラスを継承し2つのタイマーを作成せよ。1つはプログレスバーのスタイルで残り時間を表示するもの、もう1つはなにか適当なものを考えて作成せよ。