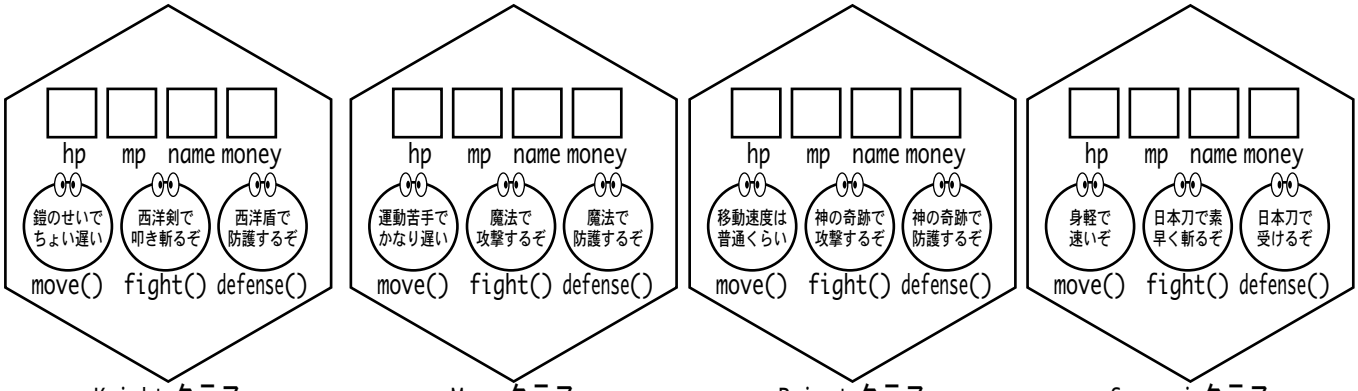
■継承ドリル解説編 改訂版　　学籍番号　　氏名

継承について授業で学習した要点をまとめてみよう。

【1】プログラムでは「だいたい同じ性質と振る舞いを持っているが，細かい振る舞いは違う」というクラスが多数必要になる。

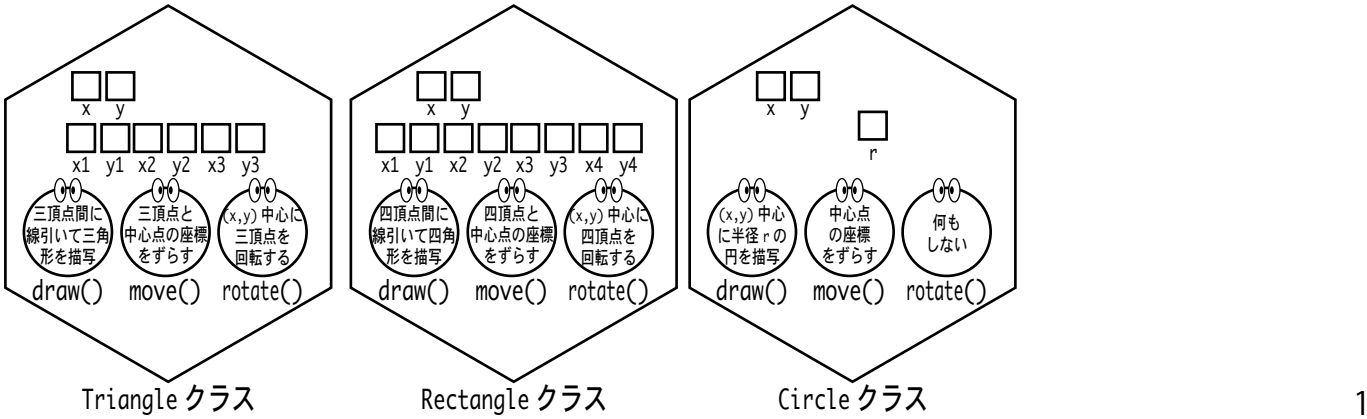
例 1) RPG ゲームのキャラクタを表すクラス - 括弧の中は共通

騎士のキャラクターを表す Knight （HP,MP, 名前 , 所持金 , 移動する , 攻撃する，防御する）
魔法使いのキャラクターを表す Mage （HP,MP, 名前 , 所持金 , 移動する , 攻撃する，防御する）
僧侶のキャラクターを表す Priest （HP,MP, 名前 , 所持金 , 移動する , 攻撃する，防御する）
侍のキャラクターを表す Samurai （HP,MP, 名前 , 所持金 , 移動する , 攻撃する，防御する）
などなど。
※これらは
・HP,MP, 名前 , 所持金 (これらは情報なのでフィールドで表す) という共通の性質と，
・移動する , 攻撃する，防御する，(これらは処理なのでメソッドで表す) という共通の動作
を持っている。(ただし，騎士は騎士らしく移動する , 攻撃する，防御するし，魔法使いは魔法使いは魔法使いらしく移動する , 攻撃する，防御する。つまり，振る舞いに関しては同じ移動する , 攻撃する，防御すると言ってもクラス毎に移動の仕方，攻撃の仕方，防御の仕方は異なる)



例 2) 製図ソフトの図形を表すクラス - 括弧の中は共通

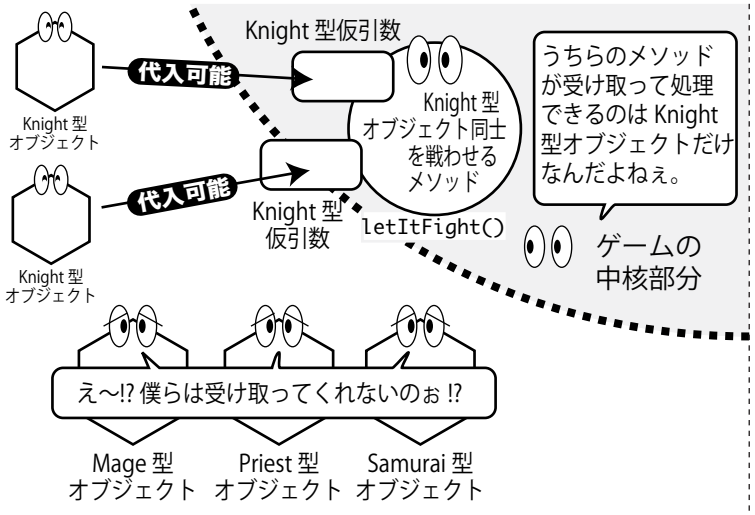
三角形という部品を表す Triangle (中心点の x 座標 , 中心点の y 座標 , 描画する，移動する , 回転する)
四角形という部品を表す Rectangle (中心点の x 座標 , 中心点の y 座標 , 描画する，移動する , 回転する)
円という部品を表す Triangle (中心点の x 座標 , 中心点の y 座標 , 描画する，移動する , 回転する)
などなど。
※これらは
・中心点の x 座標 , 中心点の y 座標 (これらは情報なのでフィールドで表す) という共通の性質と，
・描画する，移動する , 回転する，(これらは処理なのでメソッドで表す) という共通の動作
を持っている。(ただし，三角形は自分自身を三角形として描画するし，円は自分自身を円として描写する。つまり，振る舞いに関しては同じ描画する，移動する , 回転すると言ってもクラス毎に描写の仕方，移動の仕方，回転の仕方は異なる)



【2】ところが，これらのクラスのオブジェクトをソフトウェアの他の部分で受け取って利用しようとすると，困ったことになってくる。

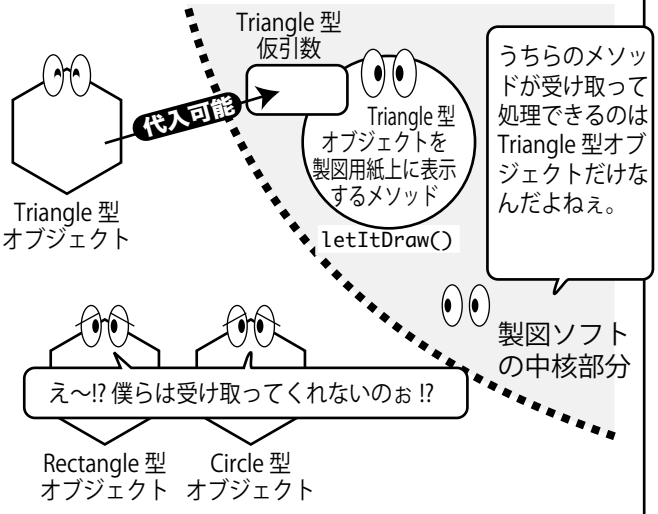
例 1) RPG ゲームの例

もしゲームの中核部分が Knight 型オブジェクトのみを扱うように書かれていたら，他の Mage 型オブジェクト ,Priest 型オブジェクト ,Samurai 型オブジェクトは扱えないことになる。



例 2) 製図ソフトの例

もし製図ソフトの中核部分が Triangle 型オブジェクトのみを扱うように書かれていたら，他の Rectangle 型オブジェクト ,Circle 型オブジェクトは扱えないことになる。



【3】この問題を解決するには，「継承 (inheritance)」を使う。Java では，既存のクラスから新しい「サブクラス」を作ることが出来る。元になるクラスのことは「スーパークラス」と呼ぶ。

```
class サブクラス名  extends  スーパークラス名 {
    新たに追加したいメンバをここに定義する
}
```

・サブクラスはスーパークラスの全メンバ (フィールド，メソッド）を受け継いでいることになる（サブクラスがスーパークラスを”継承”する)。つまり，

●サブクラスはスーパークラスと同じ性質・能力を持っている

●サブクラスから生成されたオブジェクトはスーパークラスから生成されたオブジェクトと同じ性質・能力を持っている

そのため，Java では

●サブクラスはスーパークラスとしても扱える

●サブクラスから生成されたオブジェクトはスーパークラスから生成されたオブジェクトとしても扱える

このサブクラスとスーパークラスの関係を「サブクラス is-a スーパークラス (サブクラスはスーパークラスの一種である)」といい，単に「is-a」関係とも呼ぶ。

・サブクラスでは，スーパークラスから継承したメンバ以外のメンバを新たに追加することも出来る。

・サブクラスでは，スーパークラスから継承したメソッドをサブクラス内で定義し直すことが出来る。これをメソッド・オーバーライド (method override) と呼ぶ。

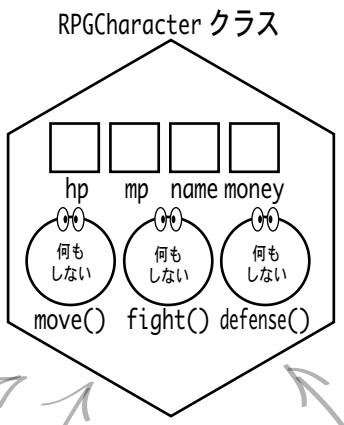
継承とメソッド・オーバーライドによって，前述の問題を解決することができる。次にその様子を見てみよう。

【4】まず，共通項のあるにかよったクラスから共通するメンバをあつめて，それらを「ひとくくりに表現する」クラスを定義する。これがスーパークラスになる。

例 1) RPG ゲームの例　Knight クラス，Mage クラス，Priest クラス，Samurai クラスから共通のメンバをあつめて，クラス RPGCharacter を作る。これがスーパークラスとなる。

※メソッドの動作内容は Knight, Mage,Priest, Samurai の各クラスで微妙に異なるので，RPGCharacter の move(), fight(), defense() の定義内容は無難なモノになっている。

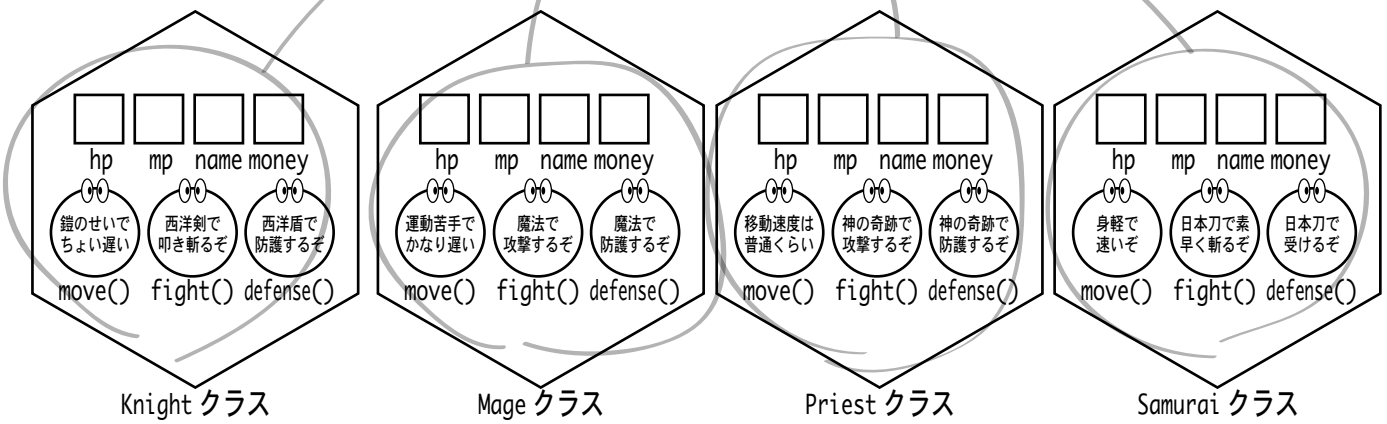
※こうして定義された RPGCharacter クラスは，「RPG に登場するキャラクター」一般をひとくくりに表している。




```
class RPGCharacter {
  private int hp, mp, money;
  private String name;
  RPGCharacter( int hp, int mp, int money, String name ){
    this.hp = hp; this.mp = mp;
    this.money = money; this.name = name;
  }
  void move( double sec ) { /* sec 秒間移動 */ }
  void fight( RPGCharacter c ) { /* c へ攻撃 */ }
  void defence( double damage ) {
    /* 受けたダメージ damage を軽減して hp から引く */
  }
}
```

※説明を簡便化するためアクセッサは省略

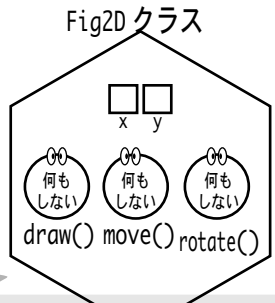
共通するメンバである hp,mp,name,money,move(),fight(),defense() を集めて，クラスを作る。



例 2) 製図ソフトの例　Triangle クラス，Rectangle クラス，Circle クラスから共通のメンバをあつめて，クラス Fig2D を作る。これがスーパークラスとなる。

※メソッドの動作内容は Triangle, Rectangle, Circle の各クラスで微妙に異なるので，Fig2D の draw(), move(), rotate() の定義内容は無難なモノになっている。

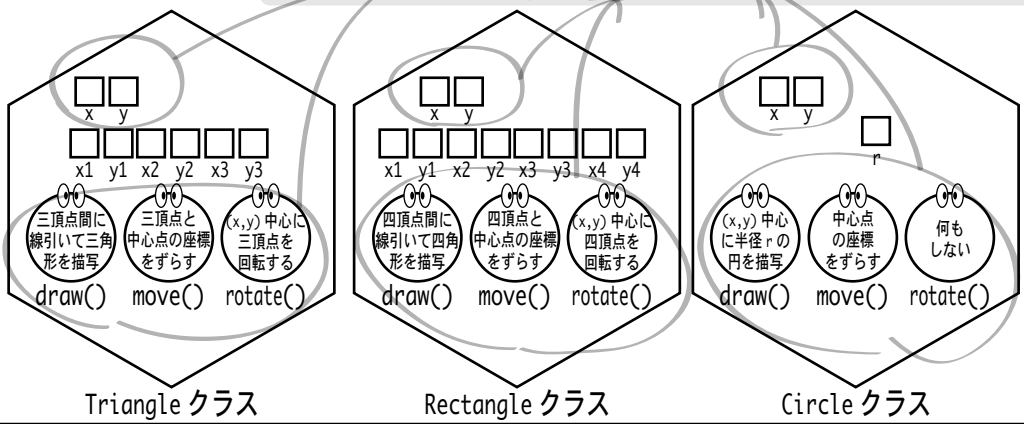
※こうして定義された Fig2D クラスは二次元図形一般をひとくくりに表している。




```
class Fig2D {
  private double x, y;
  Fig2D( double x, double y ) { this.x = x; this.y = y; }
  void draw( ) { /* 自分自身を描写する */ }
  void move( double dx, double dy ) { /* dx,dy だけ移動 */ }
  void rotate( double a ) { /* a 度だけ時計回りに回転 */ }
}
```

※説明を簡便化するためアクセッサは省略

共通するメンバである x,y,draw(),move(),rotate() を集めて，クラスを作る。



【5】スーパークラスになるべきクラスを作ったら，そこから必要なサブクラスを作る。また，
サブクラスの中でメソッドを適切にオーバライドする。

例 1) RPG ゲームの場合

RPGCharacter のサブクラスとして Knight, Mage, Priest, Samurai クラスを定義し，スーパークラスから継承したメソッドのうち必要な物 (ここでは move(), fight(), defense()) をオーバライドする。

```
class Knight extends RPGCharacter {
  Knight( int hp, int mp, int money, String name ){
    super( hp, mp, money, name ); // スーパークラスのコンストラクタ呼び出し
  }
  void move( double sec ) { sec 秒間標準より少し遅く移動 */ }
  void fight( RPGCharacter c ) {
    c を西洋剣で攻撃。攻撃が成功したら c.defense( 大きい値 );
  }
  void defence( double damage ) {
    相手から与えられたダメージ damage 鎧効果で大きく軽減して
    自分の hp から引く。
  }
}
```

```
class Mage extends RPGCharacter {
  Mage( int hp, int mp, int money, String name ){
    super( hp, mp, money, name ); // スーパークラスのコンストラクタ呼び出し
  }
  void move( double sec ) { sec 秒間標準よりかなり遅く移動 */ }
  void fight( RPGCharacter c ) {
    c を魔法で攻撃し mp を減じる。成功したら c.defense( かなり大きい値 );
  }
  void defence( double damage ) {
    魔法で防御し mp を減じる。成功したら相手から与えられたダメージ
    damage を大きく軽減して hp から引く。
  }
}
```

```
class Priest extends RPGCharacter {
  Priest( int hp, int mp, int money, String name ){
    super( hp, mp, money, name ); // スーパークラスのコンストラクタ呼び出し
  }
  void move( double sec ) { sec 秒間標準程度の速さで移動 */ }
  void fight( RPGCharacter c ) {
    c を神の奇跡で攻撃し mp を減じる。攻撃成功なら
    c.defense( そこそこ大きい値 );
  }
  void defence( double damage ) {
    神の奇跡で防御し mp を減じる。成功したら相手から与えられたダメージ
    damage をそこそこ軽減して hp から引く。
  }
}
```

```
class Samurai extends RPGCharacter {
  Samurai( int hp, int mp, int money, String name ){
    super( hp, mp, money, name ); // スーパークラスのコンストラクタ呼び出し
  }
  void move( double sec ) { sec 秒間標準よりかなり速く移動 */ }
  void fight( RPGCharacter c ) {
    c を日本刀で攻撃する。成功したら通常は c.defense( 標準的な値 );
    ごくまれに c.defense( ものすごく大きな値 );
  }
  void defence( double damage ) {
    日本刀で攻撃をはじく。失敗率高し。成功したら相手から与えられた
    ダメージ damage を大きく軽減して自分の hp から引く。
  }
}
```

例 2) 製図ソフトの場合

Fig2D のサブクラスとして Triangle, Rectangle, Circle クラスを定義し，スーパークラスから継承したメソッドのうち必要な物 (ここでは draw(), move(), rotate()) をオーバライドする。

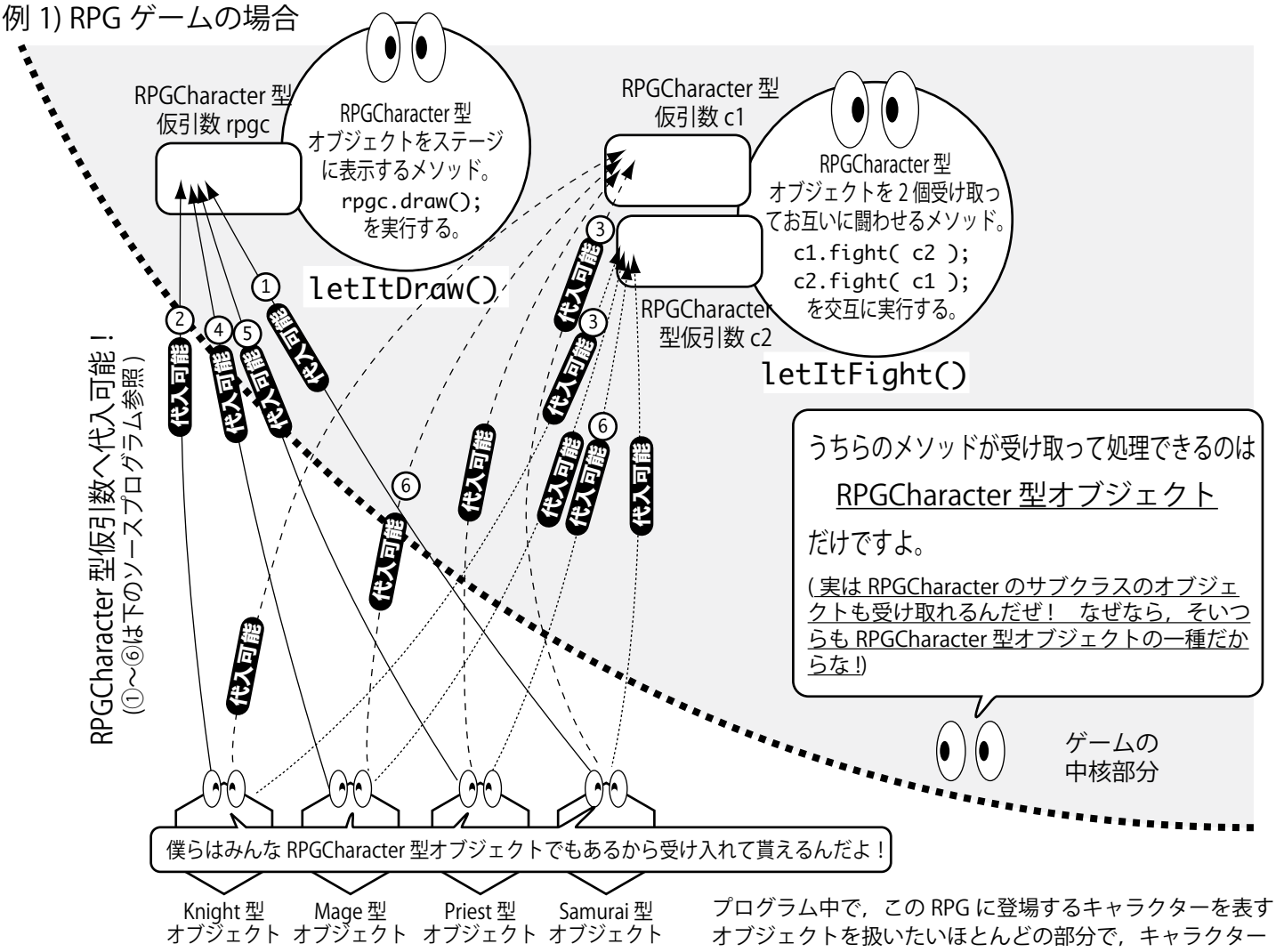
```
class Triangle extends Fig2D {
  private double x1, y1, x2, y2, x3, y3;
  Triangle( double x1, double y1, double x2, double y2, double x3, double y3 ) {
    super( (x1+x2+x3)/3.0, (y1+y2+y3)/3.0 ); // スーパークラスのコンストラクタを呼び出す
    this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2;
    this.x3 = x3; this.y3 = y3;
  }
  void draw( ) { (x1,y1),(x2,y2),(x3,y3) の 3 点間に直線を引く }
  void move( double dx, double dy ) {
    x = x + dx; y = y + dy; x1 = x1 + dx; y1 = y1 + dy;
    x2 = x2 + dx; y2 = y2 + dy; x3 = x3 + dx; y3 = y3 + dy;
  }
  void rotate( double a ) {
    (x,y) を中心に 3 頂点 (x1,y1),(x2,y2),(x3,y3) を a 度時計回りに
    回った位置の座標に更新する。
  }
}
```

```
class Rectangle extends Fig2D {
  private double x1, y1, x2, y2, x3, y3, x4, y4;
  Triangle( double x1, double y1, double x2, double y2,
            double x3, double y3, double x4, double y4 ) {
    // スーパークラスのコンストラクタを呼び出
    super( (x1+x2+x3+x4)/4.0, (y1+y2+y3+y4)/4.0 );
    this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2;
    this.x3 = x3; this.y3 = y3; this.x4 = x3; this.y3 = y3;
  }
  void draw( ) { (x1,y1),(x2,y2),(x3,y3) の 3 点間に直線を引く }
  void move( double dx, double dy ) {
    x = x + dx; y = y + dy; x1 = x1 + dx; y1 = y1 + dy;
    x2 = x2 + dx; y2 = y2 + dy; x3 = x3 + dx; y3 = y3 + dy;
    x4 = x4 + dx; y4 = y4 + dx;
  }
  void rotate( double a ) {
    (x,y) を中心に 4 頂点 (x1,y1),(x2,y2),(x3,y3),(x4,y4) を a 度時計
    回りに回った位置の座標に更新する。
  }
}
```

※サブクラス Circle の定義は省略します。
自分で定義してみて下さい。

【6】サブクラスのオブジェクトを処理する部分は，「スーパークラスのオブジェクトを受け取って処理する」ように定義する。

例 1) RPG ゲームの場合




```
class RPGSystem {

  void letItDraw( RPGCharacter rpgc ) {
    rpgc.draw( );
  }

  void letItFight( RPGCharacter c1, RPGCharacter c2 ) {
    c1.fight( c2 );
    c2.fight( c1 );
  }

  public static void main( String args[ ] ) {
    Samurai s = new Samurai( 80,    0, 100, "Ryoma" );
    Knight  k = new Knight ( 100,   0, 200, "Arthur" );
    Mage    m = new Mage   (  70, 100,  80, "Misty" );
    Priest  p = new Priest ( 100,  75, 250, "Adam"  );
    RPGSystem sys = new RPGSystem (); // RPG システム生成
    sys.letItDraw( s ); // 侍がステージに表示される①
    sys.letItDraw( k ); // 騎士がステージに表示される②
    sys.letItFight( s, k ); // 侍と騎士が闘う③
    sys.letItDraw( m ); // 魔法使いがステージに表示される④
    sys.letItDraw( p ); // 僧侶がステージに表示される⑤
    sys.letItFight( m, p ); // 魔法使いと僧侶が闘う⑥
  }
}
```

プログラム中で，この RPG に登場するキャラクターを表すオブジェクトを扱いたいほとんどの部分で，キャラクター一般を表す RPGCharacter 型を使って処理を書いておけばよいことになる (例：上図及び左のソースプログラムの灰色部分)。

こうすることで，それらの部分はサブクラスのオブジェクトを受け取り，処理することができる。

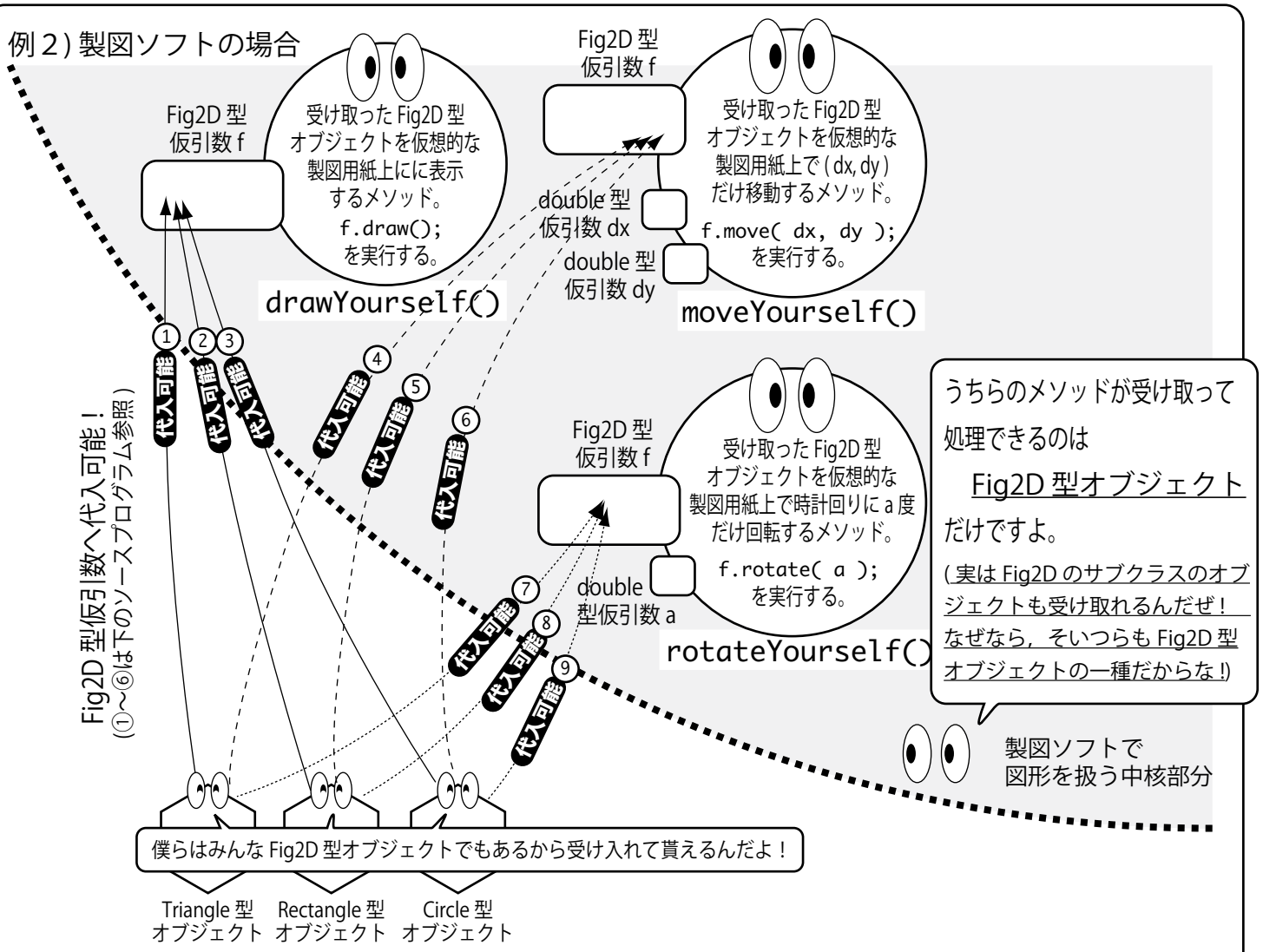
バリエーションを得たければ (例えば蛮族を表す Barbarian)，RPGCharacter のサブクラス Barbarian を新たに定義してメソッドを適切にオーバライドするだけでよい。

RPGCharacter のサブクラスがいくら増えたとしても，プログラムの殆どの部分 (例：上図および左のソースプログラムの灰色部分) は一切変更する必要がない。これは継承がもたらすオブジェクト指向の大きな美点である。

なお，この例では Knight, Mage, Priest, Samurai といったクラスを定義してその共通メンバを集めてスーパークラスを定義したが，慣れてくるとスーパークラスをいきなり定義できるようになる。

(たとえば，『RPG のキャラクターなら，騎士だろうが魔法使いだろうが僧侶だろうが侍だろうが，hp, mp, 所持金 , 名前がフィールドとして必要だし，移動したり，闘ったり，防御したりできなきゃいけないからそれぞれメソッドを用意してやる必要があるな…』と言うように。)

例２) 製図ソフトの場合




```
class DrawingPaper {

  void drawYourself( Fig2D f ) { f.draw( ); }

  void moveYourself( Fig2D f, double dx, double dy ) {
    f.move( dx, dy );
  }

  void rotateYourself( Fig2D f, double a ) {
    f.rotate( a );
  }
}

class TestDrawingPaper {
  public static void main( String args[ ] ) {
    Triangle  t = new Triangle(0.0,0.0, 1.0,0.0, 0.0,1.0);
    Rectangle r = new Rectangle(0.0,0.0, 1.0,0.0, 1.0,1.0, 0.0,1.0);
    Circle    c = new Circle( 0.0, 2.0 );
    DrawingPaper aPaper = new DrawingPaper();
    aPaper.drawYourself( t ); // 製図用紙上に三角形が表示される①
    aPaper.drawYourself( r ); // 製図用紙上に長方形が表示される②
    aPaper.drawYourself( c ); // 製図用紙上に円が表示される③
    aPaper.moveYourself( t, 1.0, 1.0 ); // 三角形を (1.0,1.0) 移動④
    aPaper.moveYourself( r, 1.0, 1.0 ); // 長方形を (1.0,1.0) 移動⑤
    aPaper.moveYourself( c, 1.0, 1.0 ); // 円を (1.0,1.0) 移動⑥
    aPaper.rotateYourself( t, 45.0 );// 三角形を 45 度時計方向回転⑦
    aPaper.rotateYourself( r, 45.0 );// 長方形を 45 度時計方向回転⑧
    aPaper.rotateYourself( c, 45.0 );// 円を 45 度時計方向回転⑨
  }
}
```

プログラム中で，この製図ソフトに登場する図形データを表すオブジェクトを扱いたいほとんどの部分で，二次元図形データー般を表す Fig2D を使って処理を書いておけばよいことになる (例：上図及び左のソースプログラムの灰色部分)。

こうすることで，それらの部分はサブクラスのオブジェクトを受け取り，処理することができる。

バリエーションを得たければ (例えば直線を表す Line)，Fig2D のサブクラス Line を新たに定義してメソッドを適切にオーバライドするだけでよい。

Fig2D のサブクラスがいくら増えたとしても，プログラムの殆どの部分 (例：上図および左のソースプログラムの灰色部分) は一切変更する必要がない。これは継承がもたらすオブジェクト指向の大きな美点である。

なお，この例では Triangle, Rectangle, Circle, といったクラスを定義して，その共通メンバを集めてスーパークラスを定義したが，慣れてくるとスーパークラスをいきなり定義できるようになる。

(たとえば，『製図ソフトで扱う図形データなら，三角形だろうが長方形だろうが円だろうが，中心座標 x, y がフィールドとして必要だし，自分自身を表示させたり，移動させたり，回転させたりできなきゃいけないから，それぞれメソッドを用意してやる必要があるな…』と言うように。)