計算機プログラミングI

第12回　2003年1月23日(木)

- インターフェース
- スレッド
- 最後に
  - ◊お知らせ
  - ◊クイズ
  - ◊授業アンケート
  - ◊実験協力のお願い

インターフェース

- (復習) クラスの継承
  - ◊親クラスにあるメソッドを子クラスは継承
    → 子クラスも同じ名前のメソッドを全て持つ
- インターフェース
  - ◊クラスの親子関係とは別に
    「同じ名前のメソッドを持っている」ことを宣言する仕組
  - ◊同じ名前のメソッドを持っている
    → 同じように操作できる

インターフェースを使ったプログラム

インターフェースの宣言

```
interface Drawable {
  void draw(int x,int y);
}
```

```
class DrawHouse extends House implements Drawable{
   int s = 10;
   public void draw(int x, int y) {
       up(); moveTo(x, y, 0); down();
              house(s);
        s+= 10;
   }
}
```

クラスがインターフェースを実装していることの宣言

```
class DrawText  implements Drawable{
   int s = 1;
   public void draw(int x, int y) {
= 0; i < x / 10; i++){
.out.print("*");

= 0; i < y / 10; i++){
.out.print("+");

ut.println("");
```

インターフェースの使用

```
public cl      81 {
   public s     ic void main(String args[]){
     Turtle rame f = new TurtleFrame();
     Drawable[] hm = new Drawable[3];
     hm[0] = new DrawHouse(); …
     …
     while(true){ …
            hm[n].draw(x, y);
```

インターフェースの働き

意味: drawという名前のメソッドがある

```
interface Drawable {
  void draw(int x,int y);
}
```

```
class DrawHouse extends House implements Drawable{
  int s = 10;
  public void draw(int x, int y) {
     up(); moveTo(x, y, 0); down();
          house(s);
     s+= 10;
  }
}
```

クラスにdrawという名前のメソッドがある

```
class DrawText  implements Drawable{
  int s = 1;
  public void draw(int x, int y) {
     for(int i = 0; i < x / 10; i++){
        System.out.print("*");
     }
     for(int i = 0; i < y / 10; i++){
        System.out.print("+");
     }
     System.out.println("");
  }
}
```

「drawという名前のメソッドがある」オブジェクトの配列

```
public class Test {
  public static void main(String args[]){
    TurtleFrame f = new TurtleFrame();
    Drawable[] hm = new Drawable[3];
    hm[0] = new DrawHouse(); …
    …
    while(true){ …
          hm[n].draw(x, y);
```

インターフェース

- 宣言:「特定の名前のメソッドを持っていること」に
  名前をつける
  メソッドの中身は無い!!!

```
interface Drawable {
  void draw(int x,int y);
}
```

- 実装: インターフェースが決めたメソッドを
  クラスが持っている

```
class DrawHouse extends House implements Drawable{
```

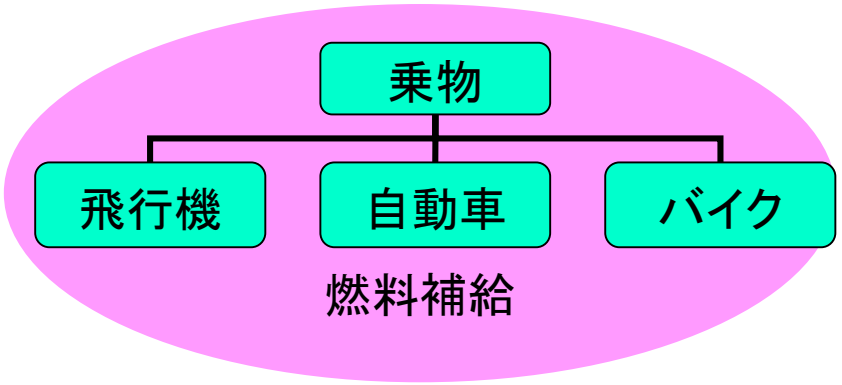
- 型としての使用: 値はそのインターフェースを
  実装したクラスのオブジェクトである

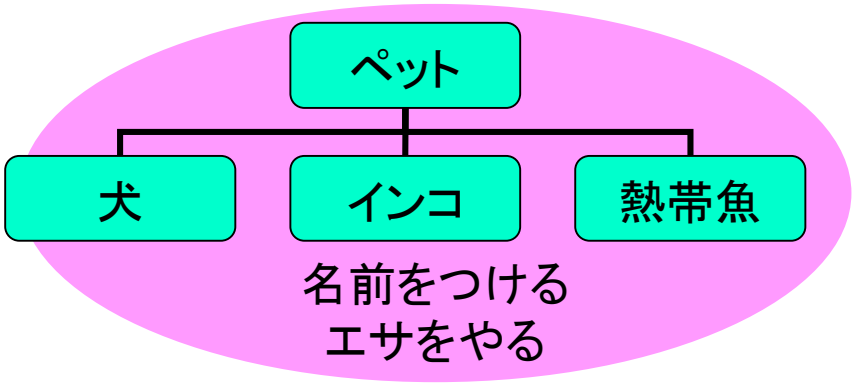
```
Drawable[] hm = new Drawable[3];
...
hm[n].draw(x, y);
```

インタフェースの目的

- クラスの異なるオブジェクトを同じように扱いたい場合がある
  - ◊ クラスの親子関係がある
    → 親クラスのオブジェクトとして扱える
  - ◊ 親子関係がない
    → インタフェースを使う

「特定のメソッドを持っている」ことは時としてクラスの親子関係と独立

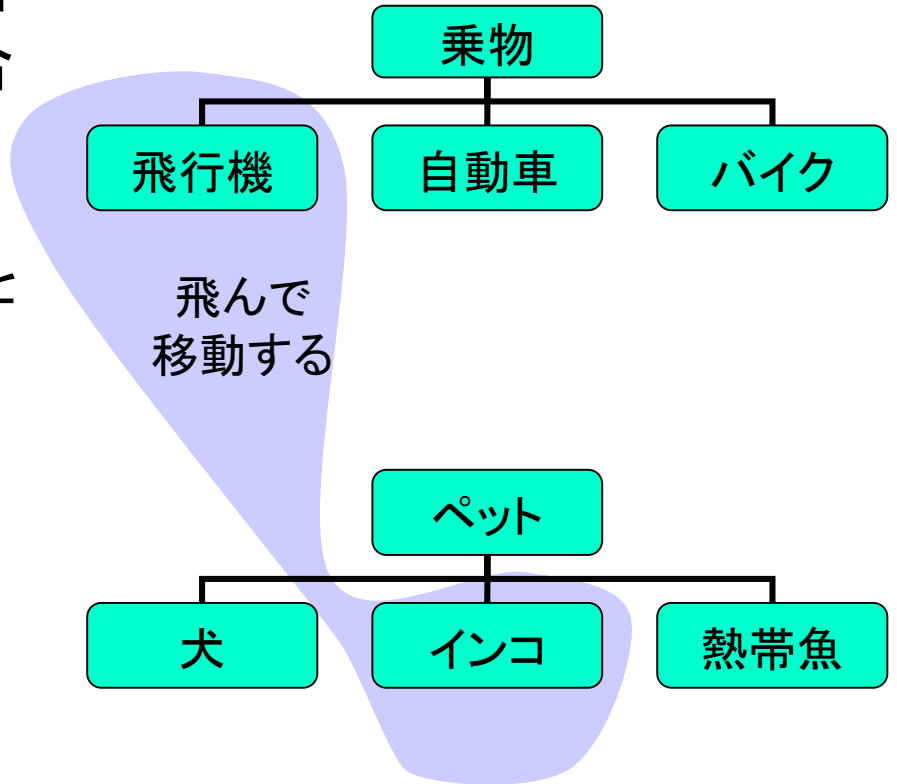




インタフェースの目的

- クラスの異なるオブジェクトを同じように扱いたい場合がある
  - ◊ クラスの親子関係がある
    → 親クラスのオブジェクトとして扱える
  - ◊ 親子関係がない
    → インタフェースを使う

「特定のメソッドを持っている」ことは時としてクラスの親子関係と独立



練習

- (準備)
  - ◊Drawable.java — インターフェースの定義
    Drawableを実装するクラスの定義
  - ◊T81.java — Drawableを使うクラス
  - もしインターフェースがなかったらどうなるか?
- 練習8.1

スレッド

- (復習) “java Sketch” のようにして、プログラムの実行を開始すると、Sketchクラスのmainメソッドの先頭、次の場所・・・のように順に処理が始まる。メソッド呼び出しがあれば、そのメソッドの先頭、次の場所・・・、最後の場所を処理し、メソッド呼び出しの次の場所から処理を続ける。

- スレッド = この「場所」を順にたどるもの
  「処理の流れ」
- これまではスレッドは1つの実行につき1つだけ
- 実は複数作ることができる

スレッドが1つのプログラムの実行

ListDemo

```
public class ListDemo {
    public static void main(String[] args) {
        f.addButton("追加");
        f.addButton("削除");
        f.addButton("前進");
        f.addButton("回転");
        TurtleList l = null;
        while (true) {
            String command = f.getPressedButton();
            if (command.equals("追加")) {
                int x = f.getClickedX("クリックして下さい");
                int y = f.getClickedY();
                Turtle m = new Turtle(x,y,0);
                f.add(m);
                m.fd(0);
                l = new TurtleList(m,l);
            } else if (command.equals("前進")) {
                l.forwardAll(10);

            } else if (command.equals("回転")) {
                Turtle.speedAll(Turtle.speedFast);
                l.turnAll();
                Turtle.speedAll(Turtle.speedSlow);
            } else if (command.equals("削除")){
                Turtle m = l.firstTurtle;
                m.kameColor = java.awt.Color.red;
                m.fd(0);
                l = l.nextTurtles;
        }
    }
}
```

TurtleList

```
public class TurtleList {
    public Turtle     firstTurtle;
    public TurtleList nextTurtles;
    public TurtleList(Turtle f, TurtleList n) {
        firstTurtle = f;
        nextTurtles = n;
    }
    public void forwardAll(int s) {
        firstTurtle.fd(s);
        if (nextTurtles != null) {
            nextTurtles.forwardAll(s);
        }
    }
    public void turnAll() {
        firstTurtle.lt((int)(Math.random()*360));
        if (nextTurtles != null) {
            nextTurtles.turnAll();
        }
    }
}
```

Turtle

```
public class Turtle{
  boolean withKame = true;
  public static boolean withKameAll = true;
  public Color kameColor = Color.green;
  TurtlePanel f;  // set by TurtlePanel
  double angle;         // turtle current angle
  double x, y;          // turtle current position
  double dx, dy;        // dx = sin(angle),  dy = -cos(angle)
  boolean penDown;      // pen status (up or down)
  Color c = Color.black; // pen color
  int kameType = 0;
  int rx, ry;
  boolean rubber = false;
  int moveWait = 20;
  int rotateWait = 20;
  public Turtle(int x,int y, int ia)
  {
    this.x = ((double)x + 0.5);
    this.y = ((double)y + 0.5);
    setangle((double)ia *Math.PI/180.0);
    penDown = true;
  }
  public double kameScale = 0.4;
  public int kame[][] = kameFig;
  public int kameR[][] = kameRFig;
  public int kameL[][] = kameLFig;
  void kameDraw(Graphics g, int data[][]) {
    int ix = (int)x, iy = (int)y;
    g.setColor(kameColor);
    for (int i = 0; i < data.length; i++) {
      int px = 0, py = 0;
      for (int j = 0; j < data[i].length; j += 2) {
        int kx = data[i][j], ky = data[i][j+1];
        int nx = (int)((kx*(-dy) + ky*(-dx)) * kameScale);
        int ny = (int)((kx*dx+ ky*(-dy)) * kameScale);
        if (j > 0) g.drawLine(ix + px, iy + py, ix + nx, iy + ny);
        px = nx;
        py = ny;
      }
    }
    g.setColor(c);
    g.fillOval(ix - 1, iy - 1, 2,2);
  }
  void show(Graphics g)
  {

    if (rubber) {
      g.setColor(c);
      g.drawLine(rx, ry, (int)x, (int)y);
    }
    if (withKame && withKameAll) {
      switch ((kameType/2) % 4) {
      case 0: case 2:
        kameDraw(g, kame);
        break;
      case 1:
```

スレッドを使ったプログラム

```
public static void main(String[] args){
  TurtleFrame f = new TurtleFrame();
  Multi91 m = new Multi91(200
  f.add(m);
  Multi91 m1 = new Multi91(10
  f.add(m1);
  m1.setColor(new Color(255,0,0));
  m1.speed(5);
  Thread t = new Thread(m);
  Thread t1 = new Thread(m1);
  t.start();
  t1.start();
  System.out.println("Main メソッドは終了する.");
}
```

```
import java.awt.Color;
public class Multi91 ext
                impleme
 int n, s;
 Multi91(int x, int y, int a, int n, int s){
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(){
     polygon(n,s);
   }
}
```

```
import java.awt.Color;
public class Multi
                im
 int n, s;
 Multi91(int x, int y, int a, in
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(
     polygon(n,s);
   }
}
```

mのrunから処理を始める
スレッドをそれぞれ作る

スレッドの処理を
開始させる

スレッドを使ったプログラム

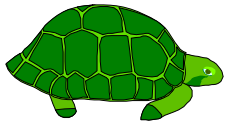
最初のスレッドは
mainから始まる

```
public static void main(String[] args){
   TurtleFrame f = new TurtleFrame();
   Multi91 m = new Multi91(200
   f.add(m);
   Multi91 m1 = new Multi91(10
   f.add(m1);
   m1.setColor(new Color(255,0,0));
   m1.speed(5);
   Thread t = new Thread(m);
   Thread t1 = new Thread(m1);
   t.start();
   t1.start();
   System.out.println("Main メソッドは終了する.");
}
```

```
import java.awt.Color;
public class Multi91 ext
                  impleme
 int n, s;
 Multi91(int x, int y, int a, int n, int s){
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(){
     polygon(n,s);
   }
}
```

```
import java.awt.Color;
public class Multi
                  im
 int n, s;
 Multi91(int x, int y, int a, in
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(
     polygon(n,s);
   }
}
```

スレッドを使ったプログラム



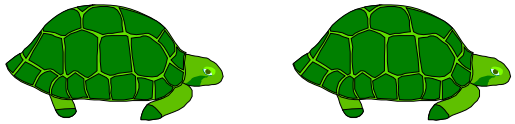
最初のスレッドは
mainから始まる

```
public static void main(String[] args){
   TurtleFrame f = new TurtleFrame();
   Multi91 m = new Multi91(200
   f.add(m);
   Multi91 m1 = new Multi91(10
   f.add(m1);
   m1.setColor(new Color(255,0,0));
   m1.speed(5);
   Thread t = new Thread(m);
   Thread t1 = new Thread(m1);
   t.start();
   t1.start();
   System.out.println("Main メソッドは終了する.");
 }
```

```
import java.awt.Color;
public class Multi91 ext
                impleme
  int n, s;
  Multi91(int x, int y, int a, int n, int s){
    super(x, y, a);
    this.n = n;
    this.s = s;
  }
    public void run(){
      polygon(n,s);
    }
}
```

```
import java.awt.Color;
public class Multi
                im
  int n, s;
  Multi91(int x, int y, int a, in
    super(x, y, a);
    this.n = n;
    this.s = s;
  }
    public void run(
      polygon(n,s);
    }
}
```

スレッドを使ったプログラム



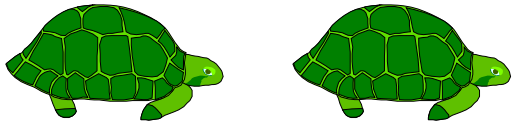
最初のスレッドは
mainから始まる

```
public static void main(String[] args){
  TurtleFrame f = new TurtleFrame();
  Multi91 m = new Multi91(200
  f.add(m);
  Multi91 m1 = new Multi91(10
  f.add(m1);
  m1.setColor(new Color(255,0,0));
  m1.speed(5);
  Thread t = new Thread(m);
  Thread t1 = new Thread(m1);
  t.start();
  t1.start();
  System.out.println("Main メソッドは終了する.");
}
```

```
import java.awt.Color;
public class Multi91 ext
                impleme
 int n, s;
 Multi91(int x, int y, int a, int n, int s){
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(){
     polygon(n,s);
   }
}
```

```
import java.awt.Color;
public class Multi
                im
 int n, s;
 Multi91(int x, int y, int a, in
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
   public void run(
     polygon(n,s);
   }
}
```

スレッドを使ったプログラム



```
public static void main(String[] args){
   TurtleFrame f = new TurtleFrame();
    Multi91 m = new Multi91(200
   f.add(m);
    ulti91 m1 = new Multi91(10
   f.add(m1);
   m1.setColor(new Color(255,0,0));
   m1.speed(5);
    Thread t = new Thread(m);
    Thread t1 = new Thread(m1);
    t.start();
    t1.start();
   System.out.println("Main メソッドは終了する.");
}
```

```
import java.awt.Color;
public class Multi91 ext
                impleme
  int n
  Mult                    ){
    sup
    this
    this
  }
  blic void run(){
     polygon(n,s);
  }
}
```

新しいスレッドは
引数のオブジェクトの
runメソッドから

```
import java.awt.Color;
public class Multi
  int
  Mu
    su
    th
    th
  }
  public void run(
     polygon(n,s);
  }
}
```

同じメソッドを
複数のスレッドが
実行することも
できる

startは
スレッドを動かし始める

スレッドを使ったプログラム

- スレッドが作られる
- 引数のオブジェクトの runメソッドが出発点になる

```
public static void main(String[]
  TurtleFrame f = new TurtleFr
  Multi91 m = new lti91(200
  f.add(m);
  Multi91 m1 = new Multi91(10
  f.add(m1);
  m1.setColor(new Color(255,0,0));
  m1.speed(5);
  Thread t = new Thread(m);
  Thread t1 = new Thread(m1);
  t.start();
  t1.start();
  System.out.println("Main メソッドは終了する.");
}
```

```
import java.awt.Color;
public class Multi91 extends House
                    implements Runnable{
 int n, s;
 Multi91(int x, int y, int a, int n
   super(x, y, a);
   this.n = n;
   this.s = s;
 }
  public void run(){
    polygon(n,s);
  }
}
```

```
interface Runnable {
   void run();
}
```

runメソッドがあるクラスの
オブジェクトであれば何でもよい
→ Runnableインターフェースを
実装したクラスの
オブジェクトであればよい

練習

- Multi91.java を動かしてみる
- タートルをもう1匹増やす
- 練習問題9.1
  - ◊タートルの配列を作る(cf. T51.java)
  - ◊スレッドの配列を作る—スレッドのi番目は
    タートルのi番目を担当
  - ◊スレッドのi番目をstart( ) — run メソッドが
    i 番目を動かす

マルチスレッドによるプログラム

- マルチスレッド = 複数のスレッド
- 利点
  - ◊別々の処理を別々にプログラムして同時に実行できる
  - ◊例:
    - データを送っている間、アニメーションを表示
    - 銀行ATMで複数の端末の操作を同時に処理
- 難点
  - ◊複数のスレッドの同期
  - ◊動きを合わせる・同じデータを操作する等々

マルチスレッドの問題の例

1つの口座に2つのスレッドが同時に預金した場合:

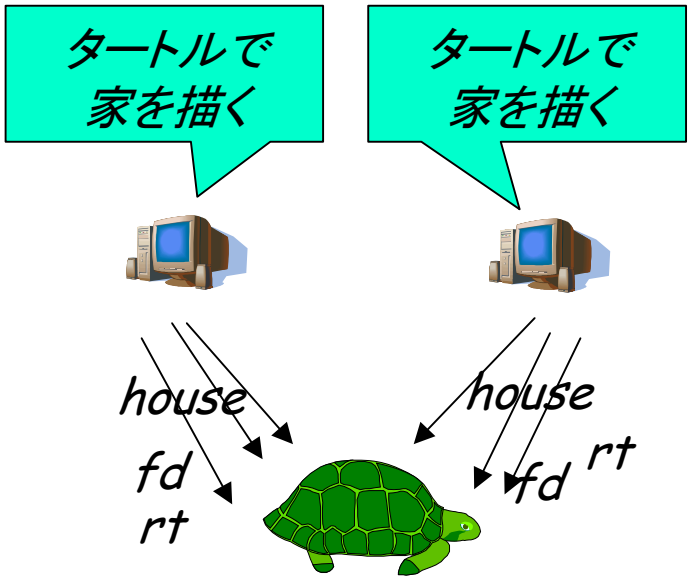
```
class 銀行口座 {
  int 残高 = ...;
  void 預金(int 預金高) {
     int 新しい残高 = 残高 + 預金高;
     預金高 = 新しい残高;
  }
}
```

スレッド1
預金(1000)
新しい残高=残高+預金高
預金高=新しい残高

スレッド2
預金(2000)
新しい残高=残高+預金高
預金高=新しい残高

マルチスレッドの処理

- 他のスレッドの実行が終了するのを待つ
  t.join()
- 他のスレッドを一時停止、再開、強制終了
- 自分が動いている間は他のスレッドを実行させない
- お知らせを待つ・お知らせをする



練習

- 練習9.3
  - ◊ Thread91.javaをそのまま実行
  - ◊ 手順に従う

  ※ “synchorized修飾子をつける”
  public synchronized void polygon(int s, int n) {
  意味: あるスレッドがこのオブジェクトのpolygonメソッドを実行している間は、他のスレッドは同じオブジェクトのpolygon(または別のsynchorized)メソッドを実行できない
- 練習9.4「2つのスレッドがsyncro()を呼び出すまで待ち、揃ったら実行を再開する」

  ※ try { 文; } catch (InterruptedException e) { } **
  「文」の中で「例外」が起きた場合、その例外を無視して**以下から実行を続ける。
  「例外」 — 突発的な事態が起きたときに、処理を中断する仕組

お知らせ

- 課題レポートの提出状況
  - ◊webページで公開・電子メールで問合せ
- ACM国際大学対抗プログラミングコンテスト
  - ◊(ala 数学オリンピック)
  - ◊対象:学部学生3名のチーム
  - ◊日程(2002年度):
    - 10月頃国内予選 @ web
    - 11月頃アジア地区予選 @ 金沢など
    - 3月頃世界大会 @ ハリウッド
  - ◊http://www.kitnet.jp/icpc/j/

- クイズ
- 実験協力のお願い
- 授業アンケート