計算機プログラミングI

第8回　2002年12月5日(木)

- メソッドとクラス　　(教科書6章)
  - ◊ インスタンスメソッド
  - ◊ インスタンス変数
  - ◊ 多重定義
- クイズ

例題: タートルでn角形を描く

- いままでTurtleは前進(fd)・回転(lt)等の指示を受けることができた

メソッド呼出し

  - ◊《監督》がfd, lt などのメソッドを順に呼出す(e.g., m.fd(10);)
  - ◊Turtleオブジェクトは言われた通りに動く

- もしタートルが「n角形を描く」という指示を受けられるなら
  - ◊《監督》は1回メソッドを呼出す (e.g., m.polygon(10,5);)
  - ◊亀が適当な順に動いてn角形を描く

例題: タートルでn角形を描く

- 亀に「n角形を描く」という指示を
  受けられるようにする
  → インスタンスメソッドを定義

- 「n角形を描」ける亀を簡単に作る
  → Turtle クラスを拡張

n角形の描き方を知っている タートルがいたら・・・

```
public class T61 {
  public static void main(String[] args){
    TurtleFrame f = new TurtleFrame( );
    House m = new House( );
    int s = 50;
    f.add(m);
    m.house(s);
    m.up( ); m.fd(s * 2); m.down( );
    m.polygon(3, s / 2);
    m.up( ); m.fd(s); m.down( );
    m.polygon(10, s / 5);
  }
}
```

House・・・Turtleのかわりのクラス名

Turtleオブジェクトと同じように使える

Turtleにはないメソッドもある

n角形の描き方を知っている タートルがいたら・・・

```
public class T61 {
  public static void main(String[] args){
    TurtleFrame f = new TurtleFrame( );
    House m = new House( );
    int s = 50;
    f.add(m);
    m.house(s);
    m.up( ); m.fd(s * 2); m.down( );
    m.polygon(3, s / 2);
    m.up( ); m.fd(s); m.down( );
    m.polygon(10, s / 5);
  }
}
```

House・・・Turtleのかわりのクラス名

Turtleオブジェクトと同じように使える

Turtleにはないメソッドもある

n角形の描き方を知っている タートルがいたら・・・

```
public class T61 {
  public static void main(String[] args){
    TurtleFrame f = new TurtleFrame( );
    House m = new House( );
    int s = 50;
    f.add(m);
    m.house(s);
    m.up( ); m.fd(s * 2); m.down( );
    m.polygon(3, s / 2);
    m.up( ); m.fd(s); m.down( );
    m.polygon(10, s / 5);
  }
}
```

House・・・Turtleのかわりのクラス名

Turtleオブジェクトと同じように使える

Turtleにはないメソッドもある

n角形の描き方を知っている タートルがいたら・・・

```
public class T61 {
  public static void main(String[] args){
    TurtleFrame f = new TurtleFrame( );
    House m = new House( );
    int s = 50;
    f.add(m);
    m.house(s);
    m.up( ); m.fd(s * 2); m.down( );
    m.polygon(3, s / 2);
    m.up( ); m.fd(s); m.down( );
    m.polygon(10, s / 5);
  }
}
```

House・・・Turtleのかわりのクラス名

Turtleオブジェクトと同じように使える

Turtleにはないメソッドもある

n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

Turtleに機能を追加したクラスを作ると宣言

追加される機能(メソッド)

自分自身を使ってn角形を描く

自分自身を使って家の形を描く

n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

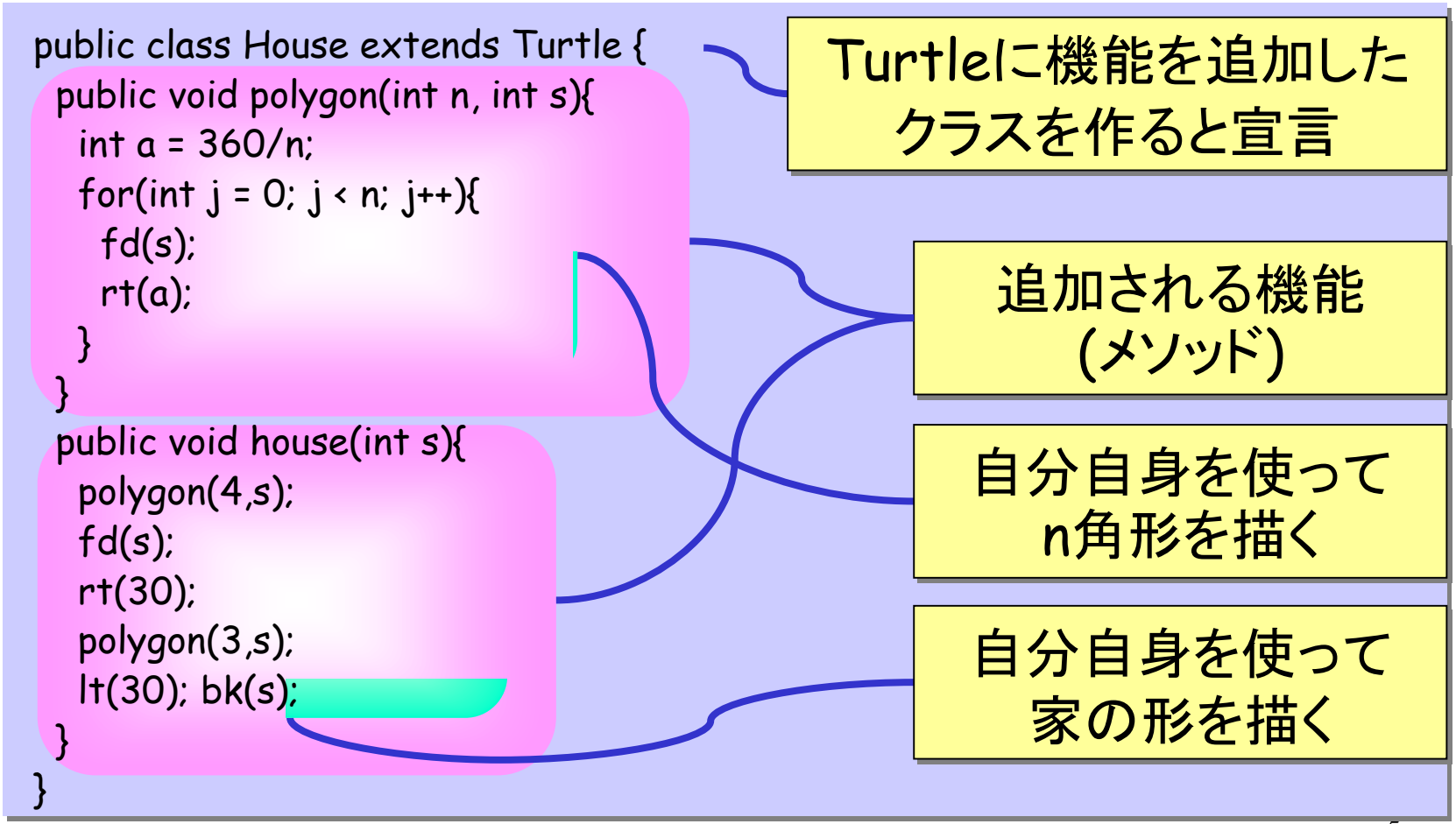
Turtleに機能を追加したクラスを作ると宣言

追加される機能(メソッド)

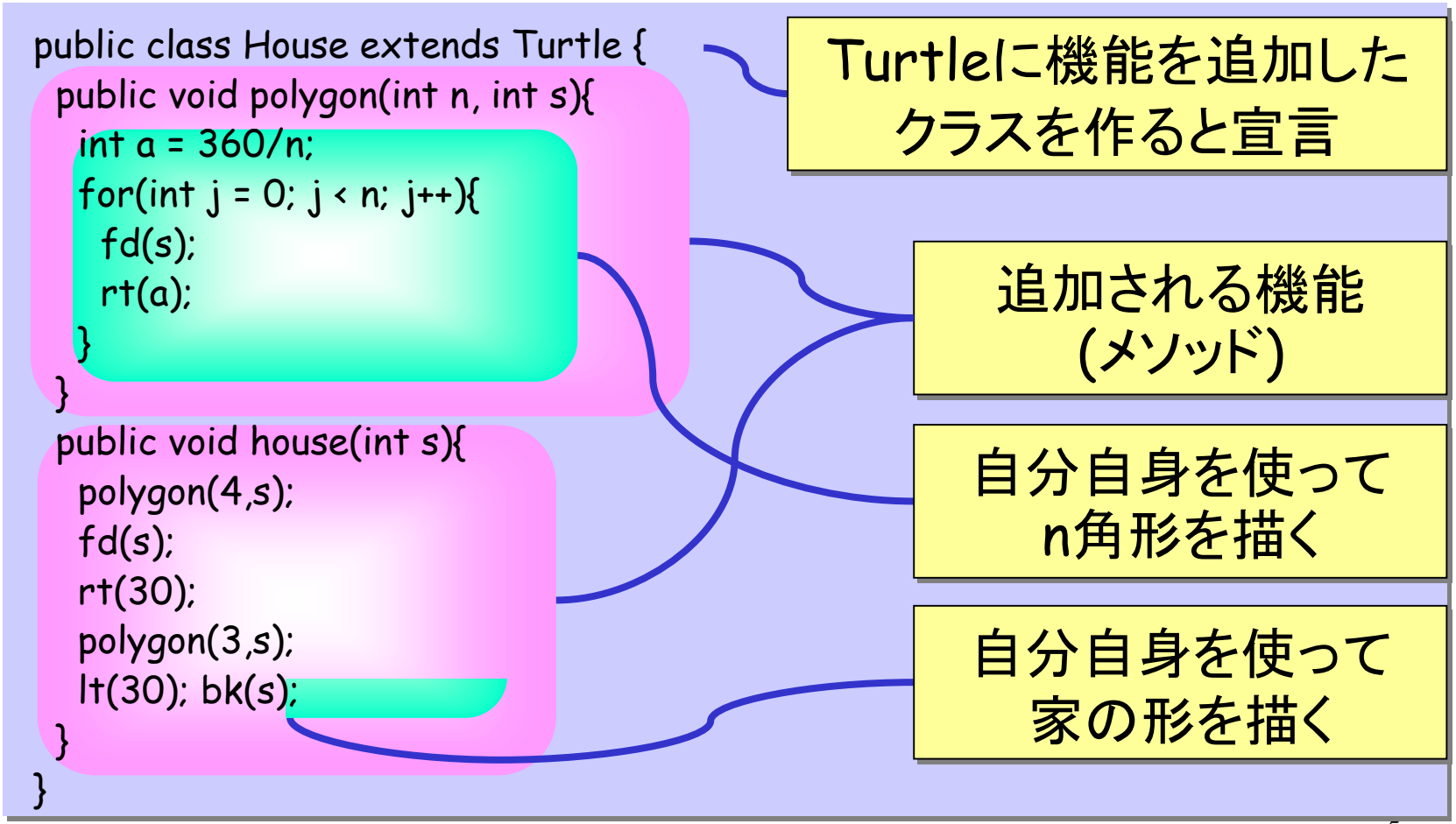
自分自身を使ってn角形を描く

自分自身を使って家の形を描く

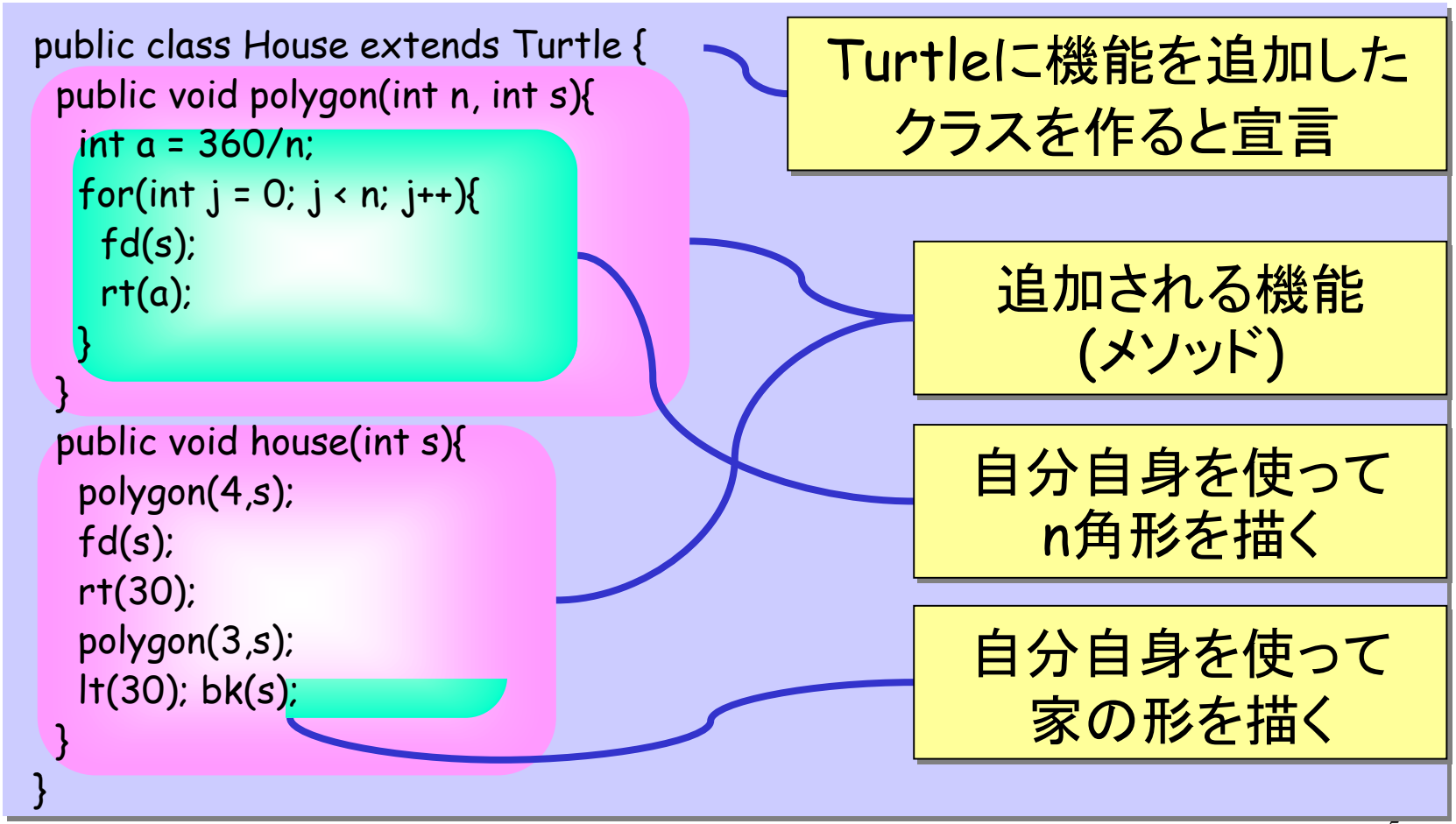
n角形の描き方を知っている タートルの定義



n角形の描き方を知っている タートルの定義



n角形の描き方を知っている タートルの定義



n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

例:m.polygon(50,3)を実行
→mに対して「多角形を描け」
→ このメソッドが実行

自分自身(=m)に対する
メソッドの実行
→m.fd(s)と同じ
→ mが前進

自分自身(=m)に対する
メソッドの実行
追加したメソッドも実行可

n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

例:m.polygon(50,3)を実行
→mに対して「多角形を描け」
→ このメソッドが実行

自分自身(=m)に対する
メソッドの実行
→m.fd(s)と同じ
→ mが前進

自分自身(=m)に対する
メソッドの実行
追加したメソッドも実行可

n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

例:m.polygon(50,3)を実行
→mに対して「多角形を描け」
→ このメソッドが実行

自分自身(=m)に対する
メソッドの実行
→m.fd(s)と同じ
→ mが前進

自分自身(=m)に対する
メソッドの実行
追加したメソッドも実行可

n角形の描き方を知っている タートルの定義

```
public class House extends Turtle {
  public void polygon(int n, int s){
    int a = 360/n;
    for(int j = 0; j < n; j++){
      fd(s);
      rt(a);
    }
  }
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
}
```

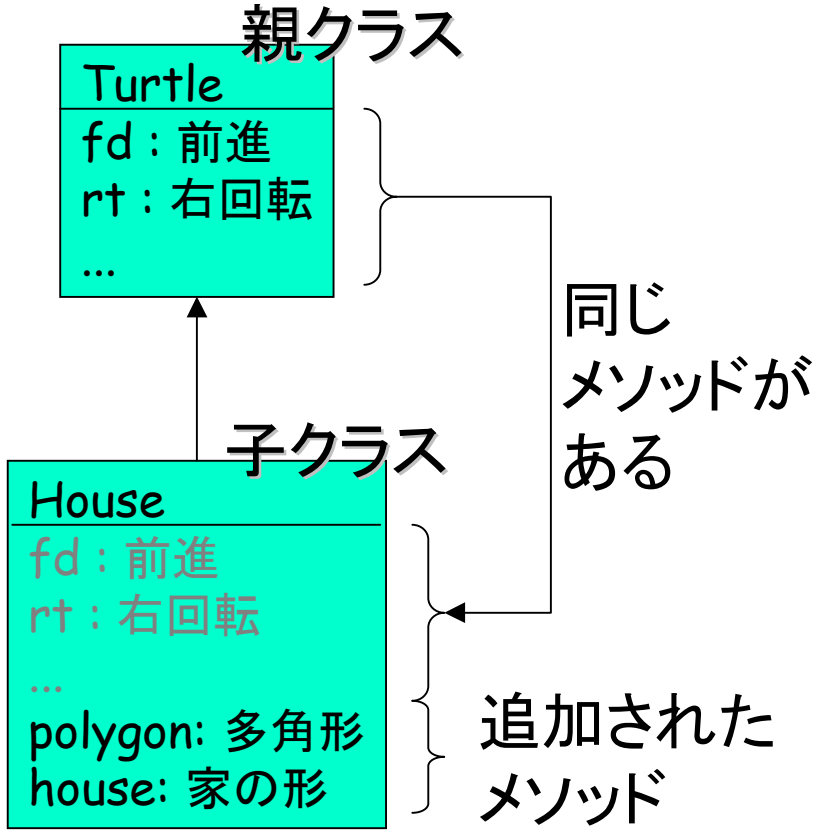
例:m.polygon(50,3)を実行
→mに対して「多角形を描け」
→ このメソッドが実行

自分自身(=m)に対する
メソッドの実行
→m.fd(s)と同じ
→ mが前進

自分自身(=m)に対する
メソッドの実行
追加したメソッドも実行可

クラスの親子関係とメソッドの継承

- class House extends Turtle {
  ...
  - ◊ House:子クラス
    or サブクラス
  - ◊ Turtle: 親クラス
    or スーパークラス
- 子クラスは親クラスにある全てのメソッドを継承
  (インスタンス変数も)
- (APIページ)



練習

準備*: House.javaとT61.javaを
コンパイル・実行

6.1*

- Houseクラスを拡張したMyTurtle クラスを作る (ヒント: Turtleクラスを拡張したのがHouseクラス)
- 「MyTurtleはメソッドhouses(n,s,w)をもつ」
  - ◊ cf.Houseクラスはメソッドhouseをもつ
  - ◊ n回の繰り返し
    - 自身のhouseメソッドで家を描く
    - 右へs+wだけ移動 (ペン上げ ・右回転・前進・左回転・ペン下げ)
- テスト: T61.javaをもとに T611.javaを作る
  - ◊ House→MyTurtle
  - ◊ m.house(s)→m.house(3,s,10)

6.2

- MyTurtle.javaにメソッド追加
- 「n角形のまわりにm角形が1辺の長さsで配置」
  - ◊ n回のくり返し
    - m角形を描く(polygon)
    - sだけ前進
    - 360/n右回転
- T611.javaを変更
  - ◊ polygon(3,s/2) →ppolygon(3,4,s/2)
  - ◊ polygon(10,s/3) →ppolygon(10,3,s/2)

6.2 多重定義(オーバーロード)

- 継承と値の型
  - ◊子クラスのオブジェクトは
    親クラスのオブジェクトのかわりとして使える
    ← 子クラスは機能を追加しただけ
  - ◊例: TurtleFrame クラスのaddメソッド
    - 引数はTurtle型 (see API)
      つまりf.add(m)のように使うときmはTurtleオブジェクト
    - 実際にはTurtleの**子クラスHouseのオブジェクト**でもよい
  - ◊型 ≈ 値の集合 とすると、
    子クラスは親クラスの部分集合

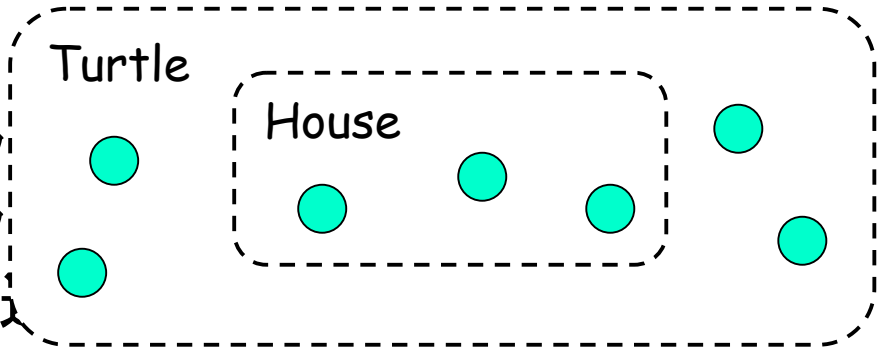
6.2 多重定義(オーバーロード)

- 継承と値の型
  - ◊子クラスのオブジェクトは
    親クラスのオブジェクトのかわりとして
    ← 子クラスは機能を追加しただけ
  - ◊例: TurtleFrame クラスのaddメソッド
    - 引数はTurtle型 (see API)
      つまりf.add(m)のように使うときmはTurtleオブジェクト
    - 実際にはTurtleの**子クラスHouseのオブジェクト**でもよい
  - ◊型 ≈ 値の集合 とすると、
    子クラスは親クラスの部分集合

```
public class T61 {
  public static void main(String[
    TurtleFrame f = new TurtleF
    House m = new House( );
    int s = 50;
    f.add(m);
    ...
```

6.2 多重定義(オーバーロード)

- 継承と値の型
  - ◊子クラスのオブジェク
    親クラスのオブジェク
    ← 子クラスは機能を



  - ◊例: TurtleFrame クラスのaddメソッド
    - 引数はTurtle型 (see API)
      つまりf.add(m)のように使うときmはTurtleオブジェクト
    - 実際にはTurtleの**子クラスHouseのオブジェクト**でもよい
  - ◊型 ≈ 値の集合 とすると、
    子クラスは親クラスの部分集合

6.2 多重定義(オーバーロード)

- 多重定義
  - ◊ 同じ名前で違うメソッド定義を用意すること
  - ◊ 引数の数や型が違っていることが必要
  - ◊ 引数の数と型で選ばれる
- シグネチャ
  - ◊ メソッドの名前+引数の型
- 例
  - ◊ houseメソッドの多重定義
  - ◊ void house(int)
  - ◊ void house(int,int,int)

```
public class House extends Turtle {
  public void house(int s){
    polygon(4,s);
    fd(s);
    rt(30);
    polygon(3,s);
    lt(30); bk(s);
  }
  public void house(int x, int y, int s) {
    up();
    moveTo(x,y,0);
    down();
    house(s);
  }
}
```

6.3 値を返すメソッド

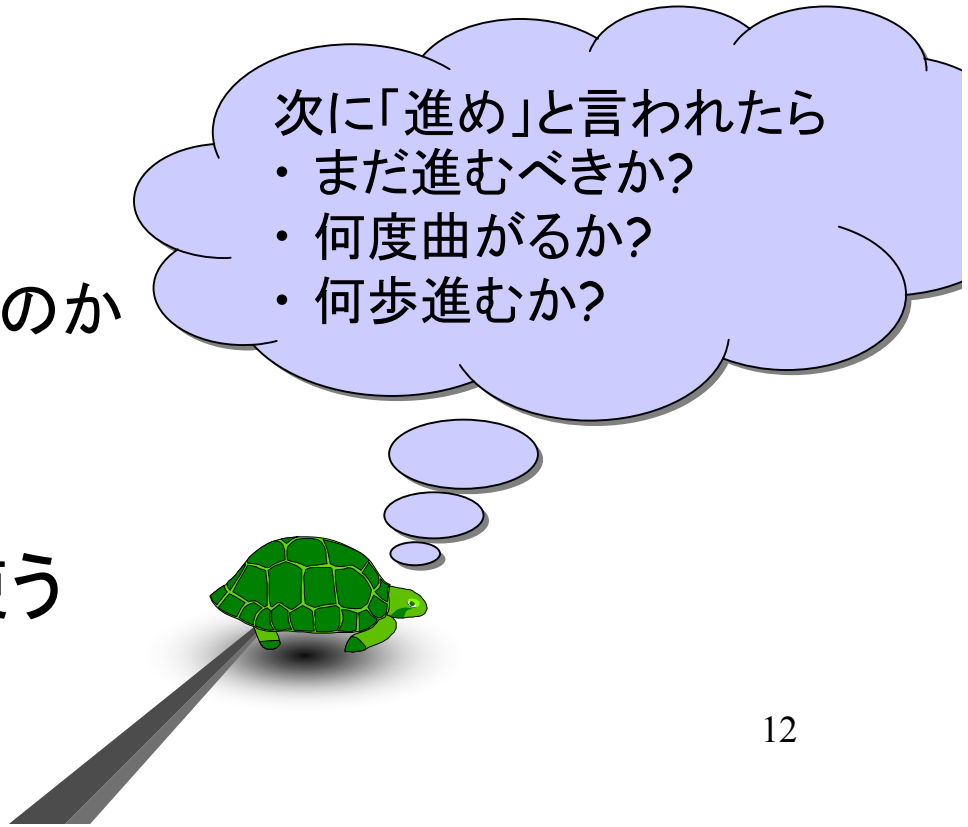
(説明済み)

例題: n角形を分割して描く亀

- 「分割して描く」とは:
  - ◊「1ステップ進め」と言われると、n角形の1辺だけを描く
- 自分がいま
  - ◊何角形を描いているのか
  - ◊1辺の長さ
  - ◊何番目の辺を描いているのか

  を覚える必要がある

→ インスタンス変数を使う



6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
…
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;
  public int step()
  {
    if(j >= n)
       return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
       return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

インスタンス変数の宣言

自分がいま
- 何角形を描いているのか
- 1辺の長さ
- 何番目の辺を描いているのか

を覚える

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

j=j+1;
if(j==n)
と同じ

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.4 インスタンス変数

```
public class Stepper ex
  public int n;
  public int size;
  private int j = 0;
  public static final int
  public static final int
  public static final int
  public int step()
  {
    if(j >= n)
        return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
        return JUST_FIN;
    return NOT_FIN;
  }
  public void reset() { j = 0; }
}
```

```
public class T62{
  public static void main(String[] args){
    TurtleFrame f = new TurtleFrame();
    Stepper m1 = new Stepper(); f.add(m1);
    Stepper m2 = new Stepper(); f.add(m2);
    m1.n = 4;   m1.size = 100;
    m2.n = 3;   m2.size = 100;
    ...
```

```
j=j+1;
if(j==n) ...
と同じ
```

いるのか

いているのか

- ローカル変数と同様に
  - 値をとり出す
  - 値をしまう

ことができる

- オブジェクトごとに用意される

→ 前回のメソッド実行時に
しまわれた値が
そのまま残っている

6.5 コンストラクタ

- オブジェクトが作られたときに実行されるメソッド
- オブジェクトのインスタンス変数を初期化などを定義しておく
- 多重定義がよく使われる
- 文法

```
クラス名(引数の型 引数の名前, ...) {
...
}
```

- コンストラクタの中から他のコンストラクタを実行できる
  - ◊ 親クラスのコンストラクタを実行する式
    super(引数, 引数, ...);
  - ◊ 自身の別のコンストラクタを実行する式
    this(引数, 引数, ...);

コンストラクタの実例

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;

  Stepper(int x, int y, int angle, int n, int size)  {
      super(x, y, angle);
      this.n = n;
      this.size = size;
    }
...
 public int step() {
   if(j >= n)
      return ALREADY_FIN;
   ...
```

```
Stepper[] hm = new Stepper[10];
for(int i = 0 ; i < 10; i++){
  hm[i] = new Stepper(i * 50 + 25,150,0, n[i], size[i]);
  hm[i].setColor(c[i % c.length]);
  f.add(hm[i]);
}
```

コンストラクタの実例

コンストラクタの定義

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;

  Stepper(int x, int y, int angle, int n, int size)  {
      super(x, y, angle);
      this.n = n;
      this.size = size;
    }
...
 public int step() {
   if(j >= n)
      return ALREADY_FIN;
   ...
```

```
Stepper[] hm = new Stepper[10];
for(int i = 0 ; i < 10; i++){
  hm[i] = new Stepper(i * 50 + 25,150,0, n[i], size[i]);
  hm[i].setColor(c[i % c.length]);
  f.add(hm[i]);
}
```

コンストラクタの実例

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;

  Stepper(int x, int y, int angle, int n, int size)  {
      super(x, y, angle);
      this.n = n;
      this.size = size;
    }
...
 public int step() {
   if(j >= n)
      return ALREADY_FIN;
   ...
```

コンストラクタの定義

```
Stepper[] hm = new Stepper[10];
for(int i = 0 ; i < 10; i++){
  hm[i] = new Stepper(i * 50 + 25,150,0, n[i], size[i]);
  hm[i].setColor(c[i % c.length]);
  f.add(hm[i]);
}
```

コンストラクタの実例

コンストラクタの定義

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;

  Stepper(int x, int y, int angle, int n, int size)  {
      super(x, y, angle);
      this.n = n;
      this.size = size;
    }
...
 public int step() {
    if(j >= n)
       return ALREADY_FIN;
    ...
```

```
Stepper[] hm = new Stepper[10];
for(int i = 0 ; i < 10; i++){
  hm[i] = new Stepper(i * 50 + 25,150,0, n[i], size[i]);
  hm[i].setColor(c[i % c.length]);
  f.add(hm[i]);
}
```

1. Stepperオブジェクトが作られる
2. コンストラクタが実行される

コンストラクタの実例

```
public class Stepper extends Turtle{
  public int n;
  public int size;
  private int j = 0;

  Stepper(int x, int y, int angle, int n, int size)  {
     super(x, y, angle);
     this.n = n;
     this.size = size;
   }
...
 public int step() {
   if(j >= n)
      return ALREADY_FIN;
   ...
```

コンストラクタの定義

親クラスのコンストラクタを実行
... new Turtle(x,y,angle) のときと同様

```
Stepper[] hm = new Stepper[10];
for(int i = 0 ; i < 10; i++){
  hm[i] = new Stepper(i * 50 + 25,150,0, n[i], size[i]);
  hm[i].setColor(c[i % c.length]);
  f.add(hm[i]);
}
```

1. Stepperオブジェクトが作られる
2. コンストラクタが実行される

```
import java.awt.Color;

public class Stepper extends Turtle {
  public int n;
  public int size;                          インスタンス変数
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1クラス変数
  public static final int  NOT_FIN = 2;

  /** x, y という座標に angle の角度の Stepper を生成 */
  Stepper(int x, int y, int angle, int n, int size) {
     super(x, y, angle);
     this.n = n;                            コンストラクタ
     this.size = size;
   }

  /** n = 4, size = 100 の Stepper を作成 */
  Stepper(int x, int y, int angle) {
     this(x, y, angle, 4, 100);             コンストラクタ
   }

  public int step() {
   if(j >= n)
      return ALREADY_FIN;
   fd(size);
   rt(360/n);                               インスタンスメソッド
   if(++j == n)
      return JUST_FIN;
   return NOT_FIN;
  }

}
```

プログラムの構造

- { } が1つのまとまり
- クラス定義の中に
  変数・メソッド等の定義
- 順番は無関係

```
import java.awt.Color;

public class Stepper extends Turtle {
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;

  /** x, y という座標に angle の角度の Stepper を生成 */
  Stepper(int x, int y, int angle, int n, int size) {
    super(x, y, angle);
    this.n = n;
    this.size = size;
  }

  /** n = 4, size = 100 の Stepper を作成 */
  Stepper(int x, int y, int angle) {
    this(x, y, angle, 4, 100);
  }

  public int step() {
    if(j >= n)
      return ALREADY_FIN;
    fd(size);
    rt(360/n);
    if(++j == n)
      return JUST_FIN;
    return NOT_FIN;
  }
}
```

インスタンス変数

クラス変数

コンストラクタ

コンストラクタ

インスタンスメソッド

プログラムの構造

- { } が1つのまとまり
- クラス定義の中に
  変数・メソッド等の定義
- 順番は無関係

```
import java.awt.Color;

public class Stepper extends Turtle {
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;

  /** x, y という座標に angle の角度の Stepper を生成 */
  Stepper(int x, int y, int angle, int n, int size) {
     super(x, y, angle);
     this.n = n;
     this.size = size;
   }

  /** n = 4, size = 100 の Stepper を作成 */
  Stepper(int x, int y, int angle) {
     this(x, y, angle, 4, 100);
   }

  public int step() {
   if(j >= n)
      return ALREADY_FIN;
   fd(size);
   rt(360/n);
   if(++j == n)
      return JUST_FIN;
   return NOT_FIN;
  }

}
```

プログラムの構造

- インスタンス変数
  public 型 名前 = 式;
- クラス変数
  public static 型 名前 = 式;
- コンストラクタ
  クラス名(型 式, ...) {
    文...
  }
- インスタンスメソッド
  public 型 名前(型 式, ...) {
    文 ...
  }
- クラスメソッド
  public static 型 名前(型 式, ...) {
    文 ...
  }

プログラムの構造

```
import java.awt.Color;

public class Stepper extends Turtle {
  public int n;
  public int size;
  private int j = 0;
  public static final int ALREADY_FIN = 0;
  public static final int JUST_FIN = 1;
  public static final int  NOT_FIN = 2;

  /** x, y という座標に angle の角度の Stepper を生成 */
  Stepper(int x, int y, int angle, int n, int size) {
     super(x, y, angle);
     this.n = n;
     this.size = size;
   }

  /** n = 4, size = 100 の Stepper を作成 */
  Stepper(int x, int y, int angle) {
     this(x, y, angle, 4, 100);
   }

  public int step() {
   if(j >= n)
       return ALREADY_FIN;
   fd(size);
   rt(360/n);
   if(++j == n)
       return JUST_FIN;
   return NOT_FIN;
  }

}
```

- インスタンス変数
  public 型 名前 = 式;
- クラス変数
  public static 型 名前 = 式;
- コンストラクタ
  クラス名(型 式, ...) {
    文...
  }
- インスタンスメソッド
  public 型 名前(型 式, ...) {
    文 ...
  }
- クラスメソッド
  public static 型 名前(型 式, ...) {
    文 ...
  }

練習

6.3

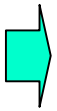
6.4 (紙に書く)

6.5 MyTurtle.html というファイルができる

準備* T62.java および Stepper.java を読み、コンパイル、実行してみる

6.6* (see リスト6.5の2行目)

```
Stepper m1 = new Stepper(); f.add(m1);
Stepper m2 = new Stepper(); f.add(m2);
m1.n = 4;  m1.size = 100;
m2.n = 3;  m2.size = 100;  m2.up();
m2.moveTo(100,200,0);  m2.down();
...
```



```
Stepper m1 = new Stepper(); f.add(m1);
Stepper m2 = new Stepper(????);
f.add(m2);
...
```

6.7

- Houseクラス → 実はある
- MyTurtleクラスに追加:

```
public MyTurtle(int x, int y, int a) {
...
}
```

6.6 クラス変数・クラスメソッド

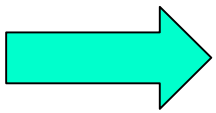
(第4回で説明済み)

立場の違い:

- **クラスメソッド**:
  「監督」の立場
  (e.g., mainメソッド)
- **インスタンスメソッド**:
  指示を受ける側の立場

メソッド=処理の中身を隠す手段
→ 誰の立場で隠しているか、の違い

オブジェクト指向の
本質の1つ



「細いことは個々の
オブジェクトが知っている」
「同じ指示を受けても、
オブジェクトによって
動きを変える」

6.7 mainメソッド
6.8 内部クラス

(略)

練習

準備: 6.9「まとめの例題」を読む

6.8

6.9 練習問題5.6参照

6.10 上で作った
void draw(int,int,int[])
を使う

6.11 draw(150,200,kameFig);などのようにする

6.12 右か左かを覚えておくインスタンス変数を1つ用意

6.13

クイズ