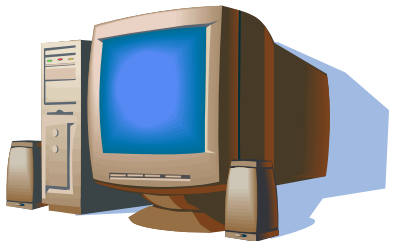
情報①

1学期　第4回

Project.4「繰り返し（ループ）」



今日の授業のながれ

- **Project.3**

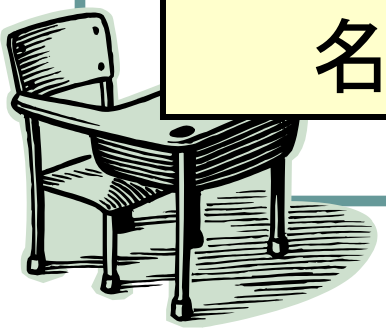
「プログラムに知性を(条件分岐、乱数)」

- **Project.4 ～**

「コンピュータの得意な仕事(繰り返し)」

  - 配布資料
    - 1学期 第4回演習チェックシート　part.1-4

クリアファイルを返却します。
名前を呼ばれたら取りに来てください。



指示をよく聞いて始める！

出欠について

- 授業ではクリアファイルと併せ、[WebClass]でも「出席」を取ります
  - 授業開始[10分後]まで⇒「出席」扱い
  - 以降[授業時間内]⇒「遅刻」扱い
  - 以降[授業終了後]⇒「欠席」扱い

- WebClass上部「出席」⇒「2025/××/××出席確認」⇒「開始」⇒「出席します！」

演習の取り組み方

演習の取り組み方

- 演習は「授業用サイト（オンラインテキスト）」を自分で読み進めて解いていく
  - 1時間目と2時間目に分けているが、自分のペースで進めること

（コンピュータ教室の場合）

- 課題の提出は「演習チェックシート」で行う

演習の取り組み方

- 例題をきちんと作ること
  - テキストは絶対に読み飛ばさない！
  - サンプルプログラムは実行してプログラムの意味を考える（納得して進む。納得できなければ，質問すること）
- 練習問題の数をこなすのが目的でない
  - 早く進むこと≠良いこと（とは限らない）
  - だらだらやること≠良いこと（とはいえない）
  - 一問だけでも，きちんと納得することが大切

演習の順番

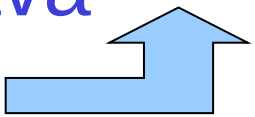
① 先にスライドでProjectの流れを掴む
② サンプルプログラムはすべて実行してプログラムの意味を理解し、例題も自分なりに一度考える
③ オンラインテキストを読む
④ 演習課題に取り組む

今日の目標

今日の目標

[Project.3]

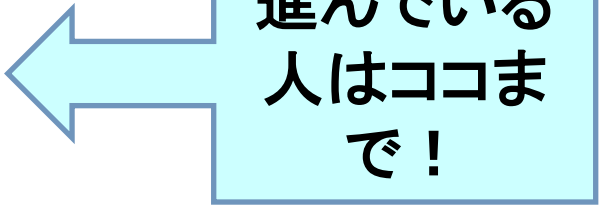
- 前回までの残り
- Kakiwake.java
- ConvertYear.java



この辺はクリアしたい

[Project.4]

- PentagonEX.java
- CircleEX.java
- Keyboard.java



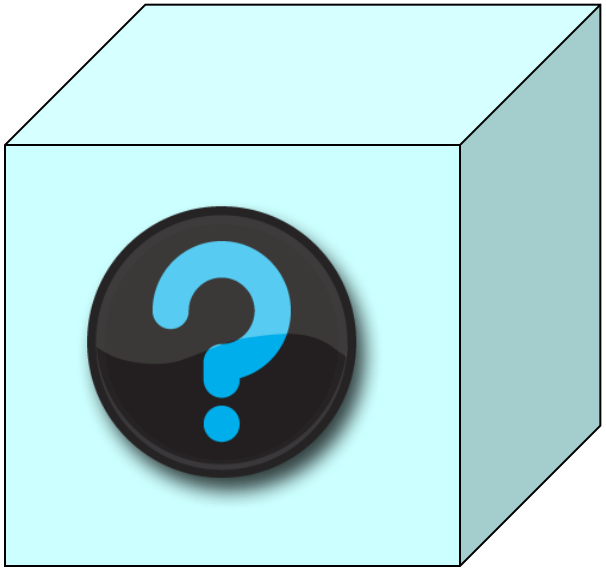
前回までの復習

Project.2
「コンピュータに情報を記憶させてみよう（変数）」

変数(前回までの復習)

- **変数**：

「数値や文字列をしまっておくための箱」



①変数名は『X』
②その値は『0』

変数　X

変数の初期化

(型名) (変数名)=(値);

int X = 0 ;など

Project.3
「条件分岐」

条件分岐とは

- ○○かどうか調べて，
  - □□だった場合は，××をする
  - △△だった場合は，★★をする
  - それ以外の場合は，■■をする

- 『ある特定の条件を満たしたときだけ実行させたい動作（あるいは、実行させたくない動作）』があるとき、条件を表す**「条件式」**を用いてプログラムの流れを制御する

条件式とは

- **『条件分岐（場合分け）』の際には〇か×か？を確実に判断できる条件が必要。**
  - 『等号・不等号（== ,<,>）』等の**比較演算子**
  - 集合関係を利用した**論理演算子**

などが条件式に用いられます。

true は 〇
false は ×
のこと

表 4.3.1.1 Javaで使える数値の比較演算子

演算子	表記例	評価
==	A==B	AとBが同じ値の時成立(true)
!=	A!=B	AとBが同じ値でない時成立(true)
>	A>B	Aが Bより大きい時成立(true)
<	A<B	Aが Bより小さい時成立(true)
>=	A>=B	Aが B以上の時成立(true)
<=	A<=B	Aが B以下の時成立(true)

表 4.3.2.1 Javaで使える論理演算子

演算子	表記例	意味
&&	A&&B	AとB両方ともtrueの時true
||	A||B	Aもしくは Bが trueの時true

①条件分岐

```
if( [条件式] ) {
    処理A
} else{
    処理B
}
```

```
int x = 1; // x の初期化

if (x == 1) {
 print("xは1です");
} else {
 print("xは1ではありません");
}
```

〇ならば

×ならば

xは1なので・・・

【実行結果】

xは1です

②条件分岐

```
if( [条件式] ) {
    処理A
}
```

```
int x = 2; // x の初期化

if (x == 1) {
 print("xは1です");
}
```

〇ならば

Xは2なので・・・

【実行結果】

（何も出力されない）

③条件分岐

```
if( [条件式①] ) {
    処理A
} else if( [条件式②] ) {
    処理B
} else{
    処理C
}
```

```
int x = 3;

if (x == 1) {
 print("xは1です");
} else if (x == 2) {
 print("xは2です");
} else {
 print("xは1か2以外です");
}
```

xは3なので・・・

【実行結果】

xは1か2以外です

【前回の補足】

コメントの書き方

if文のコメントの書き方①

- 振り分け処理の場合，ifのはじまりの括弧の後に書くとスマート

```
if (omikujiNumber == 1) {// 大吉の場合
    new ImageTurtle("daikichi.jpg");
}
```

ここに
コメント

if文のコメントの書き方②

- 特別な処理をしたい場合にif文を使う場合，目立たせるためにはじまりに書くのもあり

```
// ユーザの入力を受け取る
int input;
input = input();

// 0が入力されたら，プログラムを強制終了
if (input == 0) {
    System.exit(0);
}
```

ここにコメント

～クイズ！～

クイズ

- どんな表示になるでしょうか？

```
int n = 3;

if (n == 2) {
    print("1");
} else if (n > 2) {
    print("2");
} else if (n > 1) {
    print("3");
} else {
    print("4");
}
```

答え:「2」

～if文とif else文～

if文とif-else文の違い①

```
// おみくじを引く
int omikujiNumber;
omikujiNumber = random(4) + 1;

// おみくじの結果を表示する
if (omikujiNumber == 1) {// 大吉の場合
    new ImageTurtle("daikichi.jpg");
}

if (omikujiNumber == 2) {// 中吉の場合
    new ImageTurtle("chuukichi.jpg");
}

if (omikujiNumber == 3) {// 小吉の場合
    new ImageTurtle("shoukichi.jpg");
}

if (omikujiNumber == 4) {// 凶の場合
    new ImageTurtle("kyou.jpg");
}

update();
```

if文を羅列すれば，if-else文を使わなくても同じような動作が可能

ただし、必ず全ての場合（この場合は4回）を調べるので効率はよくない

if文とif-else文の違い2

```
// おみくじを引く
int omikujiNumber;
omikujiNumber = random(10) + 1;

// おみくじの結果を表示する
if (omikujiNumber == 1) {// 大吉の場合
    new ImageTurtle("daikichi.jpg");
} else if (omikujiNumber < 6) {// 中吉の場合
    new ImageTurtle("chuukichi.jpg");
} else if (omikujiNumber < 10) {// 小吉の場合
    new ImageTurtle("shoukichi.jpg");
} else {// 凶の場合
    new ImageTurtle("kyou.jpg");
}

update();
```

値の範囲を調べたいときはif-else文を有効活用できる

if文とif-else文の違い3

逆に値の範囲を調べたいときにif文だけだと長くなる

```
// おみくじを引く
int omikujiNumber;
omikujiNumber = random(10) + 1;

// おみくじの結果を表示する
if (omikujiNumber == 1) {// 大吉の場合
    new ImageTurtle("daikichi.jpg");
}
if (omikujiNumber > 1 && omikujiNumber < 6) {// 中吉の場合
    new ImageTurtle("chuukichi.jpg");
}
if (omikujiNumber > 6 && omikujiNumber < 10) {// 小吉の場合
    new ImageTurtle("shoukichi.jpg");
}
if (omikujiNumber == 10) {// 凶の場合
    new ImageTurtle("kyou.jpg");
}

update();
```

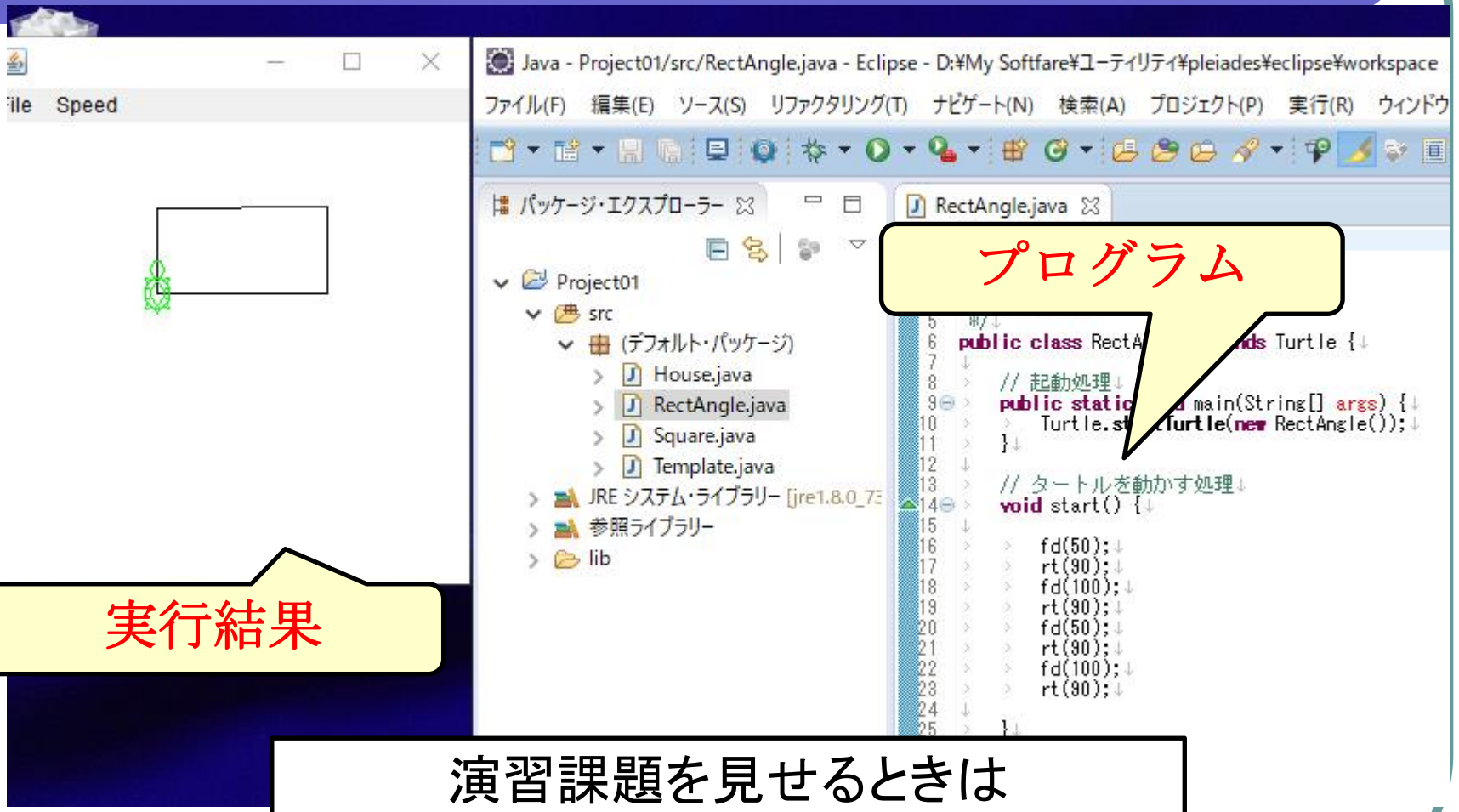
演習課題

演習について

- **演習課題の解答について**
  - 「実行結果」と「プログラム」の両方を見せる
- **課題用プログラムの作成方法**
  - Template.javaを右クリック→コピー
  - その場で右クリック→貼り付け
  - 名前の競合→(該当の演習課題の)名前を付ける

授業用サイトにコピー方法が載せました

解答例



演習課題を見せるときは

「実行結果」と「プログラム」

両方を提示すること！

演習について

- **コピー＆ペーストを利用しよう（楽をする）**
  - 今回の「Project」のプログラム例から使える場所をコピー＆ペースト
  - 今まで書いた自分のプログラムで重複する部分をコピー＆ペースト　など
- **実行結果画面サイズの変更**
  - 実行結果画面の隅を引っ張ったり、最大化する
  - window.size(640, 360);などと変更する

【基本問題】

課題 おみくじプログラム　解説

- Omikuji.javaを使って，おみくじプログラムを作る

- 大まかな処理の流れは以下のとおり
  - 1.前処理（プログラム済）
  - 2.おみくじを引く（乱数の利用）
  - 3.おみくじの結果を表示する（条件分岐、画像の表示）

課題 おみくじプログラム　解説

- 画像を表示するための命令

  例）

  ```
  new ImageTurtle(“daikichi.jpg”);
  update();
  ```

  - 画像はプログラム(Omikuji.java等)の一つ上の階層のフォルダに置く
  - 演習課題で必要な4種類の画像

  （chuukichi.jpg、daikichi.jpg、kyou.jpg、shoukichi.jpg）
  はすでにその場所に準備済み

演習問題

- **初級（OmikujiEX.java）**
  - 実行するたびに異なるおみくじの結果（画像）が表示されるようにしなさい
- **中級（OmikujiEX2.java）**
  - おみくじの結果（画像）に加えてコンソールに一言気のきいたコメントを表示させなさい（print文を使う）
- **上級（OmikujiEX3.java）**
  - 大吉，中吉，小吉，凶がでる確率を10%，40%，40%，10%になるように調整しなさい

演習の取り組み方

- 早く終わった人は・・・

  【発展問題】にチャレンジ！

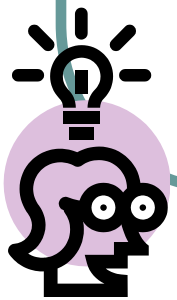
- どうしても分からない人は・・・

  次のページ【ヒント】を参照

ヒント！！！

演習問題　ヒント

- **初級（OmikujiEX.java）**
  - Janken.javaの応用
- **中級（OmikujiEX2.java）**
  - Rohrer.javaの応用
- **上級（OmikujiEX3.java）**
  - Rohrer2.javaの応用



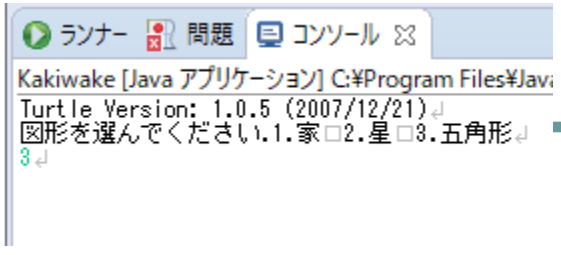
【発展問題】

練習問題

- 絵の描き分けプログラム(Kakiwake.java)

ユーザの入力に応じて、タートルが書く絵（1.家と2.星と3.五角形）を変えるプログラムを作りなさい

- **大まかな処理の流れ**
- 「1:家、2:星、3; 五角形 を描きます。数字を入力してください」と表示
- ユーザが1 or 2 or 3を入力
- 該当の図形を表示
- ※その他が入力された場合は「無効な値です」と表示

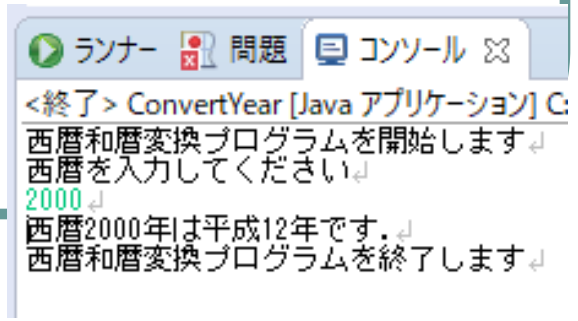


練習問題

- 西暦和暦変換プログラム(ConvertYear.java)

ユーザが西暦を入力すると、変換処理を行い、和暦を出力するプログラムを作りなさい

- **大まかな処理の流れ**
- ①「西暦を入力してください」と表示
- ② ユーザが西暦を入力
- ③「西暦xxxx年は令和○○年です」等と結果を表示
- ※和暦は「昭和(63年まで)、平成(**30**年まで)、令和」のみでその他は「無効な値です」と表示
- ※元年は1年表記



Project.4
「繰り返し（ループ）」

学習事項

学習事項

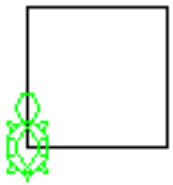
- 繰り返し
- while文
  - ループ用変数
  - インクリメント・デクリメント演算子
- 入れ子（ネスト）
  - ブロックの入れ子構造
  - 入れ子（ネスト）とループ用変数

Ⅰ．繰り返し（ループ）

繰り返し

- これまで習ってきたプログラミングでは『同じ動作』を何回かさせたい時はその回数分命令を記述する必要があった。

- ここでは前回ならった「条件式」を利用して、より簡単に図形を書くことができる「繰り返し（ループ）」について勉強する。



繰り返し

- 代表的な繰り返し構文は以下の3つ
  - while文
  - do-while文
  - for文

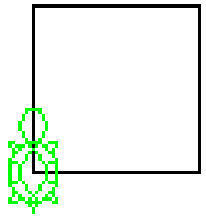
- 今回はその中でも最も扱いやすいwhile文で繰り返しを実現する方法を学習します。

繰り返し（ループ）とは

繰り返し

- 与えられた条件式を満たす限りプログラミングの特定箇所の操作が何度でも行われることを『繰り返し（ループ）』という

- 多くのプログラミング言語において使われており、プログラミングを簡潔にしたり複雑な機能を実装できる等さまざまな利点がある



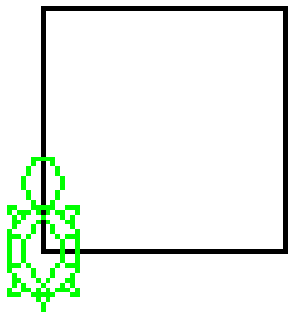
Javaの場合

- ルール：上から順番に実行される

（順次実行）

```
// タートルを動かす処理
void start() {
    fd(50);
    rt(90);
    fd(50);
    rt(90);
    fd(50);
    rt(90);
    fd(50);
    rt(90);
}
```

動かしたい回数
だけ命令を記述



Ⅱ．while文

繰り返し(ループ)

```
while（条件式）　{
        プログラミング
}
```

- 条件を満たす限り括弧の中のプログラミングの操作が何度でも繰り返し行われる

条件式について

条件式

- ○か×かを確実に判断できる式
  - 『等号・不等号(== ,<,>)』等の**比較演算子**
  - 集合関係を利用した**論理演算子**

表 4.3.1.1 Javaで使える数値の比較演算子

演算子	表記例	評価
==	A==B	AとBが同じ値の時成立(true)
!=	A!=B	AとBが同じ値でない時成立(true)
>	A>B	Aが Bより大きい時成立(true)
<	A<B	Aが Bより小さい時成立(true)
>=	A>=B	Aが B以上の時成立(true)
<=	A<=B	Aが B以下の時成立(true)

true は ○
false は ×
のこと

表 4.3.2.1 Java[illegible]

演算子	表記例	意味
&&	A&&B	AとB両方ともtrueの時true
||	A||B	Aもしくは Bが trueの時true

条件式について

- 「true」と記述することでプログラムを止めるまで無限に（チクタクと）動かし続けることもできる。

  - 例1：Circle.java

```
// 円を書く
while (true) {
  fd(1);
  rt(1);
}
```

中括弧で囲まれた部分が一行ずつ
いつまでも繰り返される

例）Circle.java
を見てみよう！

例について

① Circle.javaを実行してみる
② プログラムをよく読んでみる
③ プログラムの意味を理解する
- 「繰り返し」を使ったプログラミング
- 条件式がtrueなので、いつまでも止まらない

（※自分で×ボタンで閉じよう）

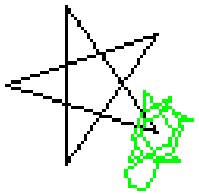
例題.1

例題：1

- StarEX.java
  - Star.java（星を描くプログラム）を繰り返しを用いたプログラムに改良しなさい

ポイント

- ①while文を使う
- ②いつまでも止まらなくてもよい
- （※自分で×ボタンで閉じよう）

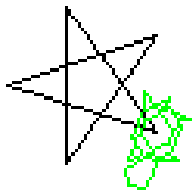


次ページの答えを見る前に自分で考えてみよう！

解答例

プログラム

File Speed



```
// タートルを動かす処理
void start() {
    while (true) {
        // 星を書く
        fd(50);
        rt(144);
    }
}
```

ループ用変数について

ループ用変数について

- 一般に繰り返しを利用する際には回数を限定して使用することの方が多い
- その場合、**ループ用の変数**を設定して以下のようにする
  - 例2：WhileHouse.java

```
int i;// ループ用変数
i = 1;
while (i <= 3) {
   fd(50);
   rt(120);
   i = i + 1;
}
```

条件を満たす限り
whileに戻って繰り返し

例) WhileHouse.java
を見てみよう！

例について

① WhileHouse.javaを実行してみる
② プログラムをよく読んでみる
③ プログラムの意味を理解する
- 「繰り返し」を使ったプログラミング
- Circle.javaとの違い
  - 条件式に「ループ用変数」を使用

例題.2

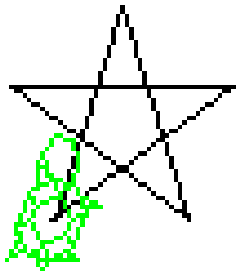
例題：2

- StarEX2.java
  - StarEx.java（星を描くプログラム）を一周したらプログラムを止めるように改良しなさい

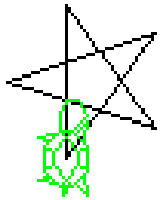
ポイント

- 条件式に「ループ用変数」を使用
- 一周したらプログラムを止めるよう改良

次ページの答えを見る前に自分で考えてみよう！



解答例



プログラム

```
// タートルを動かす処理
void start() {

    int i;// ループ用変数
    i = 1;

    while (i <= 5) {
        // 星を書く
        fd(50);
        rt(144);

        i = i + 1;
    }

}
```

インクリメント・デクリメント演算子について

インクリメント・デクリメント演算子

- 繰り返しで「ループ用変数」の値の更新「数値を1つ増加（あるいは減少）」させる処理（i = i + 1 ;など）は頻繁に登場

- 一方、「+」演算子(インクリメント演算子)や「-」演算子（デクリメント演算子）を使って簡略化して書ける

演算子	書き方	演算
＋＋	ｉ＋＋;（または、＋＋ｉ;）	i = i + 1 と同じ 『i に1 を足す』
－－	ｉ－－;（または、－－i ;）	i = i －1 と同じ 『i から1 を引く』

※ ｉ＋＋（後置）と＋＋ i（前置）は厳密には意味が異なるが、
ループ用変数の更新として用いる分には変わらない

例題.3

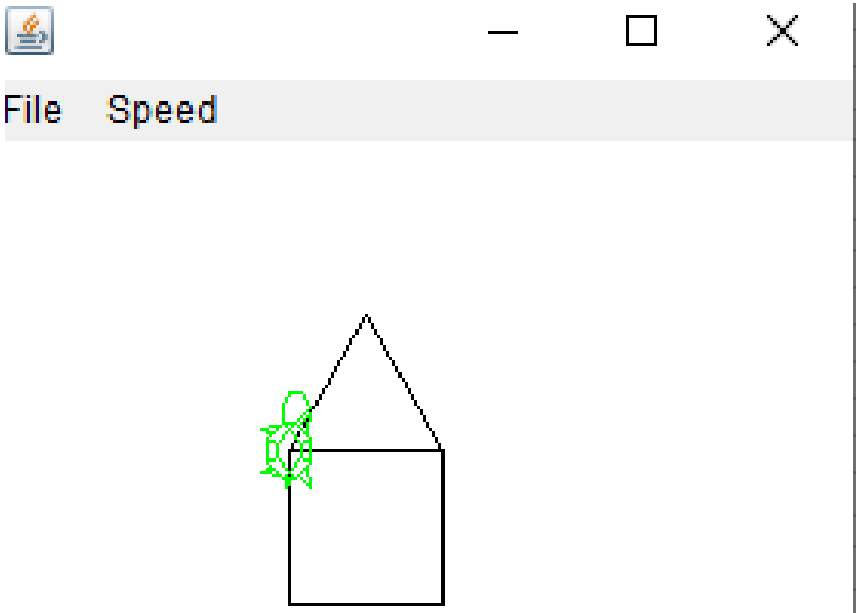
例題：3

- WhileHouseEX.java
  - WhileHouse.java（家を描くプログラム）をインクリメント演算子を用いて書き換えなさい

演算子	書き方	演算
＋＋	i ＋＋;（または、＋＋ i ;）	i = i + 1 と同じ 『i に1 を足す』
－－	i －－;（または、－－i ;）	i = i －1 と同じ 『i から1 を引く』

次ページの答えを見る前に自分で考えてみよう!

解答例



プログラム

```
// タートルを動かす
public void start() {

    // 屋根を書く
    rt(30);

    int i;// ループ用変数
    i = 1;
    while (i <= 3) {
        fd(50);
        rt(120);
        i++;
    }

    lt(30);// 上向きに戻す

    // 本体を書く
    i = 1;
    while (i <= 4) {
        rt(90);
        fd(50);
        ++i;
    }

}
```

Ⅲ.入れ子（ネスト）について

ブロックの入れ子構造

- ブロックの内側に新しいブロックを(複数)作ることを**「入れ子（ネスト）構造」**という

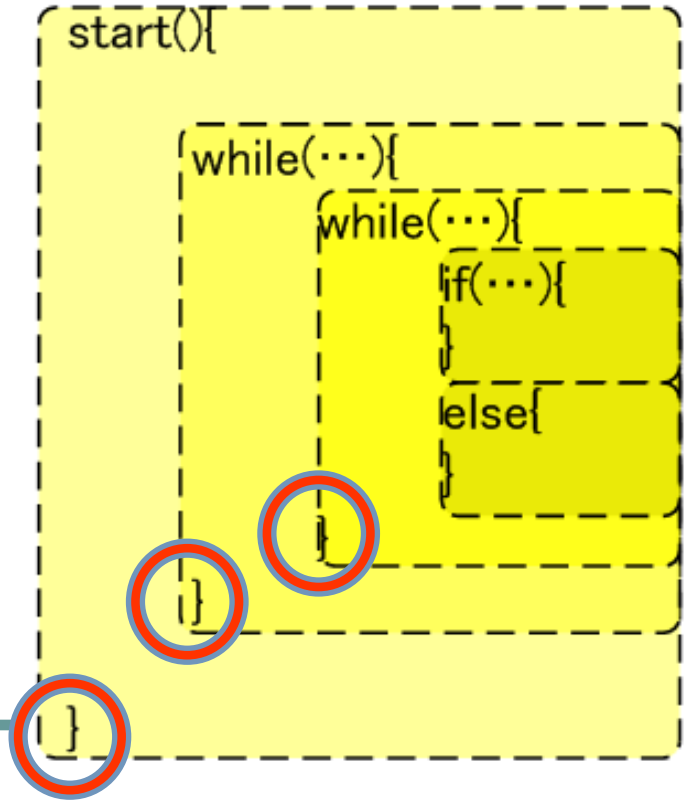
（※ブロックは中括弧({})で囲まれた部分のこと）

- 繰り返し(whileブロック)の中に繰り返し(whileブロック)たり、繰り返しに限らず条件分岐(ifブロック)を作ったりもできる

ブロックの入れ子構造

- 中カッコ{}の数が増えるので、開き括弧と閉じ括弧の数を揃えるように気を付けよう！

例)　Petal.java
Flower.java
を見てみよう！



例について

① Petal.javaを実行してみる
② 次にFlower.javaを実行してみる
③ プログラムをよく読んでみる
④ プログラムの意味を理解する
- 「入れ子」を使ったプログラミング
- 「繰り返し」を繰り返す

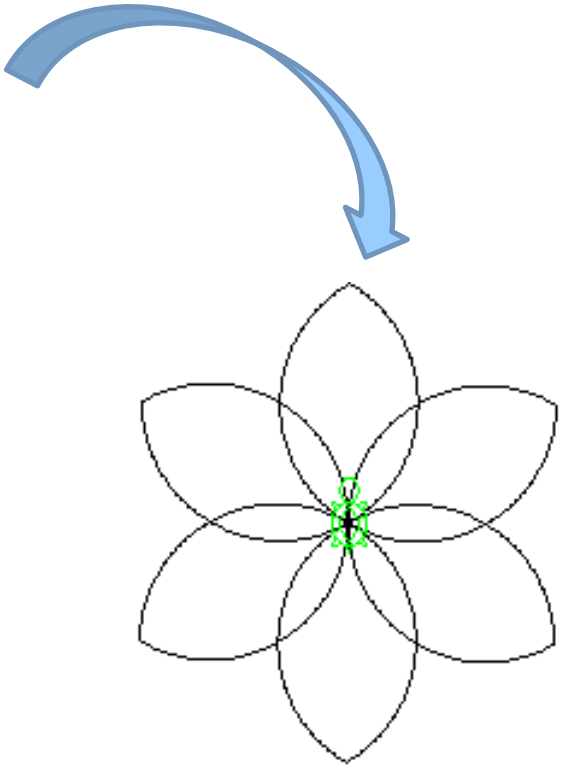
繰り返しを繰り返す

Petal.javaを（6回繰り返し）

円弧を書く（120回繰り返し）

円弧を書く（120回繰り返し）

Petal.java



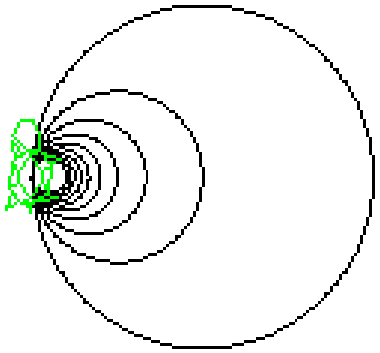
Flower.java

入れ子（ネスト）と
ループ用変数

入れ子（ネスト）とループ用変数

- 入れ子と先ほど習ったループ用変数を組み合わせるとより複雑な図形が描ける
- 特にループ用変数をループごとに変化させると・・・？

例）　MultiCircle.java
を見てみよう！



例について

① MultiCircle.javaを実行してみる
② プログラムをよく読んでみる
③ プログラムの意味を理解する
- 「入れ子（繰り返しを繰り返す）」
- 「ループ用変数」を変化

(※変数angleがループ毎に増え、図形も変化)

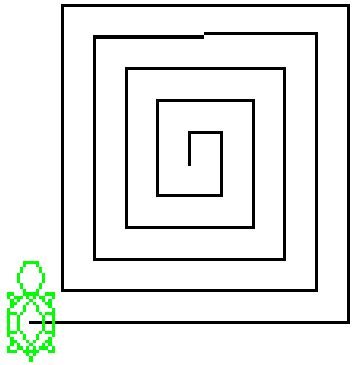
例題.4

例題：4

- Uzumaki.javaに繰り返し（while文）を利用し、右下のような渦巻を描くプログラムを作りなさい

ポイント

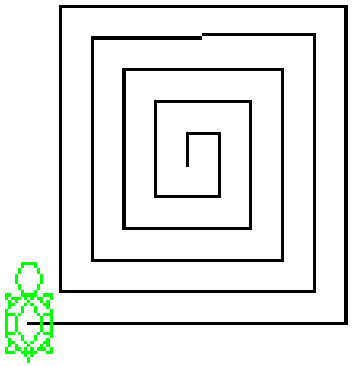
- 「入れ子（繰り返しを繰り返す）」
- 「ループ用変数」を変化



次ページの答えを見る前に自分で考えてみよう！

解答例

プログラム



```
void start() {

    // 渦巻を書くための10回ループ
    int i = 1;
    while (i <=10) {

        int j = 1;
        while (j <= 2) {
        fd(10 * i);
        rt(90);
        j++;
        }

        i++;
    }

}
```

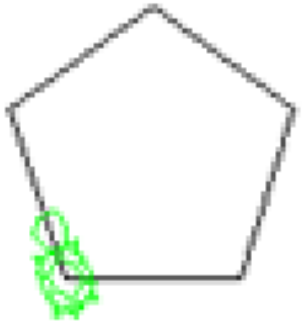
演習課題

演習課題

- PentagonEX.java
  - Pentagon.javaを改良し、繰り返し(while文)を使って正五角形を書くプログラムにしなさい

ポイント

- 一周したらプログラムを止めるよう改良

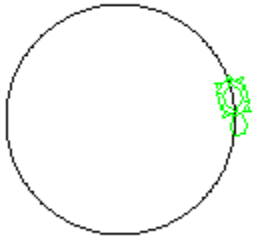


演習課題

- CircleEX.java
  - Circle.java（円を描くプログラム）を1周したら亀が止まるように改良しなさい

ポイント

- 一周したらプログラムを止めるよう改良

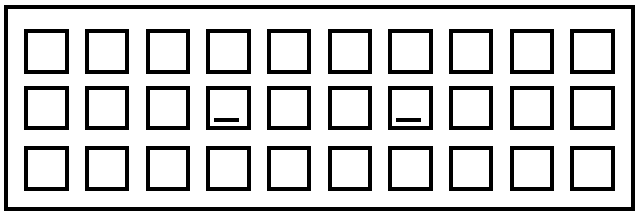


演習課題

- Keyboard.java
  - キーボードを描くプログラムを作りなさい

ポイント

- 横10列　縦3列のキー配列
  - キーの大きさ「lengthは10」キー間隔「marginは5」
- キーボードの外枠と印はプログラム済（省略）



次回予告

次回予告

- Project.4「コンピュータの得意な仕事（繰り返し）」
- Project.5「アニメーション入門(複数のオブジェクト)」

クリアファイルを
提出して帰ること！

NEXT TIME

以上

おつかれさまでした。

それでは、また次回！！