課題2 参考資料： **Java Applet**のプログラミング

この課題で使用するJava Appletのプログラミング環境について説明します．

はじめの一歩：何もしないプログラム

まずは最も簡単なプログラムを作って実行してみましょう．これは「何もしない」プログラムです．実行すると，まっさらな描画領域（以降，スクリーンと呼びます）が現れるだけです．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {

  }
}
```

このプログラムは，これから登場するすべてのプログラム（そして，あなたが課題で作成するすべてのプログラム）のひな型となります．おまじないだと思っておきましょう．このプログラムには2組の中括弧がありますが，内側の中括弧の中に，コンピュータへの命令を書いていくことになります．

実行してみましょう．実行したらブラウザの「戻る」ボタンで戻ってきて下さい．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {

  }
}
```

実行する　元の内容に戻す

×印の表示

それではスクリーンに何かを表示させるように命令してみましょう．先ほどのプログラムの内側の中括弧の中に，「g.drawBlueMark(80, 30);」という文を付け加えます．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    g.drawBlueMark(80, 30);
  }
}
```

付け加えた文の意味は「スクリーンのx座標が80，y座標が30の位置に，青い×印を表示せよ」という意味です．スクリーンの座標は左上の角が原点（x座標もy座標も0）で，原点から右に行くほどx座標が増加し，下に行くほどy座標が増加します．なお，Java Appletでは文は；（セミコロン）で終ります．忘れないように注意しましょう．

このプログラムを実行してみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    g.drawBlueMark(80, 30);
  }
}
```

実行する　元の内容に戻す

このプログラムでは，1行目と5行目以外は左端から始まっていません．これはプログラムの見やすくするためで，すべての行を左端から書き始めても文法上は問題ありません．

複数の×印の表示

コンピュータにもっとたくさん命令してみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    g.drawBlueMark(80, 30);
    g.drawBlueMark(85, 50);
    g.drawBlueMark(90, 70);
  }
}
```

このプログラムでは，複数の文が付け加えられています．コンピュータは上に書かれた文から順番に実行していきます．3つの青い×印が，それぞれ指定された位置に表示されます．

このプログラムを実行してみましょう．また，プログラムを変更して好きな個数の×印を好きな位置に表示してみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    g.drawBlueMark(80, 30);
    g.drawBlueMark(85, 50);
    g.drawBlueMark(90, 70);
  }
}
```

実行する　元の内容に戻す

変数の使用

コンピュータにもう少し複雑なこと…例えば計算…をさせてみようと思ったら，データを保存しておくための変数が必要になります．変数を使ったプログラムの例を見てみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int i = 30;
    int j = 50;
    int k;

    g.drawBlueMark(i, i);
    g.drawBlueMark(j, j);
    k = i + j;
    g.drawBlueMark(k, k);
  }
}
```

変数を使用する前に必ず 宣言をする必要があります．変数にはそれぞれ固有の名前と型（格納するデータの種類）がありますが，どんな型のどんな名前の変数を使うのかを宣言するのです．例えば，「int i = 30;」の文では，intという型（整数データを格納する型）の i という名前の変数を使用し，その初期値を30にすることを宣言しています．同様に「int j = 50;」の文で，整数データを格納する j という名前の変数を宣言し，その初期値を50としています．続く「int k;」の文では，整数データ型のkという変数を宣言していますが，kには初期値を設定していません．このように，変数の宣言には，初期値を設定する文法と，宣言だけをして初期値は設定しない文法の2通りがあります．初期値が設定されていない変数は，はじめはどんな値が入っているのか誰にもわかりません．

さて，「g.drawBlueMark(i, i);」は×印を表示させる文ですが，iには30が入っているので「g.drawBlueMark(30, 30);」と同じ意味になります．同様にjには50が入っているので，「g.drawBlueMark(j, j);」は「g.drawBlueMark(50, 50);」という意味になります．

続く「k = i + j;」の文は，変数に右辺の値を 代入する文です．右辺で計算を行っていますが，iは30，jは50ですからkには80が入ります．したがって，最後の「g.drawBlueMark(k, k);」の文は，x座標80，y座標80の位置に青い×印を表示することになります．なお，このプログラムでは最後の変数宣言（kの宣言）の次の行が空行になっていますが，これもプログラムの見やすさのためで，Java Appletではコンピュータは空行を無視します．

実行して(30, 30)，(50, 50)，(80, 80)の位置に×印が表示されるか確かめてみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int i = 30;
    int j = 50;
    int k;

    g.drawBlueMark(i, i);
    g.drawBlueMark(j, j);
    k = i + j;
    g.drawBlueMark(k, k);
  }
}
```

実行する　元の内容に戻す

スクリーンの中央

スクリーンの中央の位置に×印を表示するにはどうすれば良いでしょうか．スクリーンの大きさが分かれば，座標はコンピュータに計算させることができそうです．スクリーンの横の長さ（幅）はwidth，縦の長さ（高さ）はheightという名前の整数型の変数に格納されています．これらの変数は特別にあらかじめ定義されているものです．そこで，次のようなプログラムを書けば，スクリーンの中央に×印を表示することができます．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int x;
    int y;

    x = width / 2;
    y = height / 2;
    g.drawBlueMark(x, y);
  }
}
```

まず，スクリーンの中央に対応するx座標とy座標を表す2つの整数型の変数xとyを定義します．「x = width / 2;」の文では，変数xにスクリーンの幅を2で割った値が代入されます．例えば，スクリーンの幅（width）が500だったらxには250が代入されます．

ところで，xとwidthは整数型の変数，2ももちろん整数です．widthが奇数だったときはどうなるでしょうか．実は整数を整数で割るときには，割り切れなくても答は実数にはならず小数点以下は　　切捨てになります．つまり，widthが499だったときにはxには249が代入されます．極端な例では，1を2で割ると答は0になるのです．

実行してみましょう．ちゃんとスクリーンの中央に×印が表示されますか？

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int x;
    int y;

    x = width / 2;
    y = height / 2;
    g.drawBlueMark(x, y);
  }
}
```

実行する　元の内容に戻す

実数データ

プログラムではいつも必ず整数だけ使うとは限りません．実数のデータを扱うこともあるでしょう．実数

のデータを格納するには，intではなくdoubleという型の変数を使います．

先ほども説明したように，整数を整数で割る場合には答も整数になります．そこで，次のような場合には注意が必要です．

```
double x;

x = width / 2;
...
```

xは実数データを格納できるdoubl型で宣言されています．widthはあらかじめ整数型で定義されている特別な変数，2も整数です．右辺の割算は整数を整数で割るので小数点以下は切り捨てられます．したがって，右辺が割り切れなかった場合にも，xに代入される値は小数点以下が切り捨てられてしまいます．

除算が割り切れなかったとき，結果を実数として変数に格納したい場合には，次のようなプログラムにする必要があります．

```
double x;

x = (double)width / 2;
...
```

widthの前に(double)をつけることによって，もともと整数型のwidthの値を，特別に実数のデータとして扱うように指示します（このことをキャストするといいます）．すると右辺は実数を整数で割ることになるので，答は切り捨てられずに実数になります．xも実数を格納できるdouble型の変数として宣言されているので，問題なく代入が行われます．

計算のいろいろ

コンピュータに計算させるためには，プログラム中に計算式を書く必要があります．今までにもいくつか登場しているように，プログラムでは私達が使っている数式とほとんど同じように数式を書くことができます．しかし，いくつか注意が必要な点があります．

加算と減算には「+」と「-」を使いますが，乗算と除算には「*」と「/」の記号を使います．「(」と「)」の括弧を使って演算の順番も自由に指定できますが，中括弧や大括弧は使いません．以下の例のように，すべて「(」と「)」を用います．

```
x = ((a + b) * (c + d)) / e;
```

「xの2乗」は「x * x」と書くのが簡単です．aの平方根をxに代入するには，すこしややこしいですが，

```
x = Math.sqrt(a);
```

と書きます．もちろんこの場合のxは，実数のデータを格納できるdouble型として宣言されていなければなりません．また，

```
x = Math.abs(a);
```

と記述すると，aの絶対値がxに代入されます．

繰り返し

プログラムを書いていると，同じような処理を何度も繰り返したいことがしばしばあります．例えば一定の間隔で並んだ6つの×印を表示させるプログラムは，以下のように書けます．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    g.drawBlueMark(100, 100);
    g.drawBlueMark(120, 100);
```

```
    g.drawBlueMark(140, 100);
    g.drawBlueMark(160, 100);
    g.drawBlueMark(180, 100);
    g.drawBlueMark(200, 100);
  }
}
```

しかし，この書き方では，表示させる×印の数が増えるとプログラムを打ち込むのが大変になります．そこで登場するのが **for**ループ の構文です．

コンピュータに処理を繰り返させるには，何回繰り返したかを数えるための変数を使います．この変数のことをカウンタ変数と呼びます．forループの構文は次のようなります．

```
for (カウンタの初期化; 条件式; カウンタの更新) {
  繰り返す文1;
  繰り返す文2;
  ....
}
```

実際のforループの例と見比べながら動作を考えてみましょう．

```
for (int i = 100; i <= 200; i = i + 20) {
  g.drawBlueMark(i, 100);
}
```

まず「カウンタの初期化」の部分では，forループの繰り返しを制御するカウンタの定義と初期値の設定を行います．この例では「int i = 100」となっていますが，これは「カウンタとしてiという整数型変数を使い，その初期値を100とする」ことを指示しています．なお，ここで定義するカウンタ変数は，このforループ構文の中だけで有効になります．forループの外側からは，この変数に代入したり値を読んだりすることはできません．

カウンタの初期化が済むと，コンピュータは中括弧で囲まれた文を上から順に実行していきます．この例では「g.drawBlueMark(i, 100);」の1文だけですが，複数の文を書くことができます．iは100に初期化されていますので，はじめは(100, 100)の位置に×印を表示することになります．

中括弧で囲まれた文（ここでは1文だけですが）をすべて実行し終ると，コンピュータはカウンタの値を更新します．どのように更新するのかは「カウンタの更新」の部分に記述します．この例では「i = i + 20」となっています．「=」は「等号」ではなく「代入」を表す記号ですから，「iに20を加えた値をiに代入せよ」という意味になります．今までiは100でしたから，今度は120になるわけです．

カウンタの更新が終ると，コンピュータは中括弧の中の文をもう一度上から順に実行するかどうかを判断します．これをどう行うかを記述するのが「条件式」の部分です．この例では「i <= 200」となっています．「<=」は「以下」を表す記号なので，「iが200以下なら繰り返しなさい」という意味になります．先ほどのカウンタの更新によって現在iは120になっていますから，この繰り返しの条件を満たしています．そこで中括弧の中をもう一度実行し，今度は(120, 100)の位置に×印を表示します．

そして，中括弧の中の文をすべて実行したらカウンタを更新し，条件をチェックし…と同じことを繰り返し，カウンタの値が繰り返しの条件を満たさなくなるまで続けます．

実行する前に×印がいくつ表示されるか考えてみましょう．そして結果を確かめてみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int i = 100; i <= 200; i = i + 20) {
      g.drawBlueMark(i, 100);
    }
  }
}
```

実行する　元の内容に戻す

forループのカウンタの更新では，単にカウンタに1を加えるだけのことがよくあります．これは，

```
i = i + 1
```

と書けますが，もっと簡単に

```
i++
```

と書くこともできます．どちらも同じ意味になります．

条件式

条件式は，forループで繰り返しの条件を指定する以外にも，コンピュータにさまざまな判断をさせるのによく用います．条件式に使う「以下」や「以上」などの記号について整理しておきましょう．

```
i <= 10
```

と書くと，iが10以下（10を含む）であることを表します．

```
i < 10
```

と書くと，iが10より小さい（10は含まない）ことを表します．

```
i >= 10
```

と書くと，iが10以上（10を含む）であることを表します．

```
i > 10
```

と書くと，iが10より大きい（10は含まない）ことを表します．

```
i == 10
```

と書くと，iが10と等しいことを表します．「=」（代入の記号）と「==」は意味がまったく違いますので注意しましょう．

```
i != 10
```

と書くと，iが10と等しくないことを表します．

複雑な条件

複数の条件をつなげて複雑な条件を作ることもできます．

```
5 <= i && i <= 10
```

と書くと，iが5以上 かつiが10以下であることを表します．「&&」は「かつ」を表す記号です．

```
i == 5 || i == 10
```

と書くと，iが5に等しい または iが10に等しいことを表します．「||」は「または」を表す記号です．

条件の判定

コンピュータに何らかの判断をさせ，その判断の結果によって実行する内容を変えたい時にはif文を使います．

```
if (条件式) {
  条件が満たされた時の文;
  ...
}
else {
  条件が満たされなかった時の文;
  ...
}
```

if文では，コンピュータはまず条件式をチェックし，条件が満たされていれば1つ目の中括弧の中の文を順に実行します．そして2つ目の中括弧（elseのあと）の中の文は実行しません．逆に条件が満たされていない場合には，1つ目の中括弧の中の文はとばし，2つめの中括弧の中の文章を順に実行します．

条件を満たした時だけ何かを実行し，そうでなかったときには何も実行させないという場合には，else以降を省略して

```
if (条件式) {
  条件が満たされた時の文;
  ...
}
```

と書くこともできます．

先ほどのforループの例と組み合わせて以下のようなプログラムを考えてみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int i = 100; i <= 200; i = i + 20) {
      if (i < 150) {
        g.drawBlueMark(i, 100);
      }
      else {
        g.drawRedMark(i, 100);
      }
    }
  }
}
```

ここで新しく出てきた「g.drawRedMark(i, 100);」は赤い×印を表示させる命令文です．iが150より小さい間は青い×印を，150以上になったら赤い×印を表示します．

どのように表示されるかを予想してから実行してみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int i = 100; i <= 200; i = i + 20) {
      if (i < 150) {
        g.drawBlueMark(i, 100);
      }
      else {
        g.drawRedMark(i, 100);
      }
    }
  }
}
```

実行する　元の内容に戻す

forループの中断

forループを繰り返している途中で，気が変わって中断したくなることもたまにはあるでしょう．forループは「break;」という文に出会うと，カウンタの条件にかかわらず，その場で繰り返しの処理を終了します．例えば，次のように使うことができます．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int i = 100; i <= 200; i = i + 20) {
      if (i < 150) {
        g.drawBlueMark(i, 100);
      }
      else {
        break;
      }
    }
  }
}
```

このプログラムを実行してみましょう．なぜこのような表示になるか考えてみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int i = 100; i <= 200; i = i + 20) {
      if (i < 150) {
        g.drawBlueMark(i, 100);
      }
      else {
        break;
      }
    }
  }
}
```

実行する　元の内容に戻す

入れ子になった**for**ループの中断

Java Appletではforループの内側に，別のforループを書くことができます．このような構造のことを「　　入れ子になったforループ」といいます．入れ子になったforループでも，break文が中断するforループはbreak文が書かれたループだけで，一度に複数のループ処理を中断することはありません．

以下のプログラムは，先ほどのプログラムのbreak文つきforループの外側に，カウンタ変数jを使ったもう1つのforループを追加し入れ子のforループ構造にしたものです．この場合，break文は内側のforループ（カウンタ変数iのもの）だけを中断し，外側のループ（カウンタ変数jのもの）は中断しません．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int j = 100; j <= 200; j = j + 20) {
      for (int i = 100; i <= 200; i = i + 20) {
        if (i < 150) {
          g.drawBlueMark(i, j);
        }
        else {
          break;
        }
      }
    }
  }
}
```

このプログラムを実行してみましょう．なぜこのような表示になるか考えてみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    for (int j = 100; j <= 200; j = j + 20) {
      for (int i = 100; i <= 200; i = i + 20) {
        if (i < 150) {
          g.drawBlueMark(i, j);
        }
        else {
          break;
        }
      }
    }
  }
}
```

実行する　元の内容に戻す

forループと変数

forループを使うときには，変数の有効範囲に注意が必要です．すでに説明しましたが，以下のようにforループでカウンタ変数を定義した場合には，forループの外側からカウンタ変数の値を読むことはできません．

```
for (int i = 0; i < 10; i++) {
  ...
}
```

もしforループの処理の後にカウンタ変数の値を読みたい場合には，以下のようにforループの前にあらかじめカウンタ変数を定義しておき，forループではカウンタは定義せずに初期化だけを行います．

```
int i;

for (int = 0; i < 10; i++) {
  ...
}
```

同様にforループの中括弧の中で定義された変数も，forループの中だけで有効となります．例えば以下のような記述では，forループの処理の後で変数iの値を読むことはできますが，変数jの値を読むことはできません．

```
int i;

for (int = 0; i < 10; i++) {
  int j;
  ...
}
```

点を表示する

スクリーンに×印ではなく，他のものを表示したいこともあるでしょう．コンピュータでは図形を点の集まりとして描くので，とりあえず点が表示できれば何でも書くことができそうです．以下はスクリーンの中央に青い点を表示するプログラムです．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int x;
    int y;

    x = width / 2;
    y = height / 2;
    g.drawBluePoint(x, y);
  }
```

```
}
```

「g.drawBluePoint(x, y);」というのが，座標(x, y)の位置に青い点を表示させる命令文です．この他にも，「g.drawRedPoint(x, y);」とすれば赤い点を，「g.drawWhitePoint(x, y);」とすれば白い点を表示させることができます．

スクリーンの中央に青い点が表示されます．小さいので分かりにくいですが，よく目を凝らして確認してみましょう．

```
public class ExerciseApplet extends MyApplet {
  public void drawScreen(MyGraphics g, int width, int height) {
    int x;
    int y;

    x = width / 2;
    y = height / 2;
    g.drawBluePoint(x, y);
  }
}
```

実行する　元の内容に戻す