第2章　Javaプログラムの作成

このプリントでは、Java プログラムの開発に JDK とよばれる Sun Microsystems 社から無償で提供されるコマンドライン上の開発環境を用いる。JDK はいくつかのプログラムから成り立っているが、主に用いるのは、＿＿＿＿という Java コンパイラと ＿＿＿＿という中間コード実行プログラム（JVM エミュレータ）、それに ＿＿＿＿＿＿＿＿＿＿というアプレット（後述）を実行するためのプログラムである。

統合開発環境（IDE）としては、Sun Microsystems 社の NetBeans, IBM[1]によって開発された Eclipse[2]などがある。IDE は、エディタ・コンパイラ・デバッガなどが統合された環境で、プログラムを迅速に開発することができる。画面上でボタンなどの GUI 部品を配置することができるものもある。

この章では JDK によって Java のプログラムを作成する方法を説明する。

2.1　コンパイルと実行

新しいプログラミング言語を学習するときの慣習により、最初に、画面に “Hello World!” と表示するだけのプログラムを作成する。

2.1.1　Java アプリケーション

例題 2.1.1 まず、通常のアプリケーション（つまり後述のアプレットでない） *Hello World* プログラムは次のような形になる。このファイルを好みのエディタ（メモ帳、秀丸、*Emacs* など）で作成する。

ファイル Hello0.java

```
public class Hello0 {
  public static void main(String args[]) {
    System.out.printf("Hello World!%n");
  }
}
```

`Hello0.java` を Java 仮想機械（JVM）のコードにコンパイルするには、＿＿＿＿というコマンドを用いる。

```
> javac Hello0.java
```

コンパイルが成功すれば、同じディレクトリに `Hello0.class` というファイルができている。これが、JVM のコードが記録されているファイルである。このことを確認して、`Hello0` を ＿＿＿＿で実行する。

```
> java Hello0
```

[1]現在は IBM とは独立した組織 Eclipse Foundation で開発されている。
[2]`http://www.eclipse.org/`

（.class はつけない）すると Hello World!と画面に表示される。

javac　—　Java のソースから中間コード（JVM コード）へのコンパイラ
java　—　JVM のエミュレータ

この Hello0.java の意味を簡単に説明する。**public class** Hello0 は Hello0 という _____を作ることを宣言している。（クラスなど、オブジェクト指向の概念の詳しい説明は、後述する。ただし、Java では、どんな簡単なプログラムでもクラスにしなければならないことになっているので、とりあえずこの形を覚えておくと良い。）Java では public なクラス名（この場合 Hello0）とファイル名（この場合 Hello0.java）の .java を除いた部分は _____[3]。この例の場合はどちらも Hello0 でなければならない。この後のブレース（{）と対応する閉ブレース（}）の間がクラスの定義である。ここに変数（フィールド）や関数（メソッド）の宣言や定義を書く。

Java アプリケーションの場合も C 言語と同じように、main という名前のメソッド（関数）から実行が開始されるという約束になっている。main メソッドの型は C 言語の main 関数の型（int main(int argc, char** argv)）とは異なる void main(String args[]) という型になっている。public や static というキーワード（修飾子）については後述する。とりあえず、この形（**public static** void main(String args[])）の形のまま覚えておくと良い。

System.out.printf は C 言語の printf に相当するメソッドで文字列を書式指定に従って画面に出力する。つまりこのプログラムは、単に “Hello World!” という文字列を出力するプログラムである。%d, %c, %x, %s などの書式指定は C 言語の printf と同じように使用することができる。一方、%n は Java の書式指定に特有の書き方でシステムに依存する改行コード（Unix では ¥x0A, Windows では、¥x0D¥x0A,）である。

2.1.2　Java アプレット

例題 2.1.2 次に _____と呼ばれる *WWW* のページの中で実行される *Java* プログラムの書き方を紹介する。

次のような *2* つのファイルをエディタ（秀丸、メモ帳、*Emacs* など）で作成する。

Hello.java は、*Java* のプログラム（アプレット）のソースファイルである。

ファイル Hello.java

```
import javax.swing.*;
import java.awt.*;
  /* <applet code="Hello.class" width="150" height="25"> </applet> */

public class Hello extends JApplet {
  @ Override
  public void paint(Graphics g) {
    g.drawString("HELLO WORLD!", 50, 25);
  }
}
```

もう一つの *HelloTest.html* は、このプログラムを埋め込む方の *HTML* 文書のファイルである。アプレットを一つ表示するだけの単純な *HTML* ページである。*HTML* ファイルの名前は、クラス名や

[3] public でないクラス名に対しては、この規則は強制されないが、従っておく方が何かと便利である。

Java ファイル名と _____。（一つの *HTML* ファイルの中に複数のアプレットがある場合もあるので、これは当然である。）

ファイル *HelloTest.html*

```
<html>
<head> <title> A simple program </title> </head>
<body>
<applet code="Hello.class" width="150" height="25"> </applet>
</body>
</html>
```

Hello.java を JVM のコードにコンパイルするためには、やはり _____コマンドを用いる。

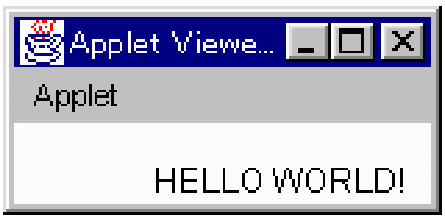
```
> javac Hello.java
```

コンパイルが成功すれば、同じディレクトリに Hello.class というファイルができている。これが、JVM のコードが記録されているファイルである。

このことを確認して、HelloTest.html を _____というプログラムで実行する。appletviewer は HTML を解釈して、その中で参照されているアプレットをテスト実行するためのプログラムである。

```
> appletviewer HelloTest.html
```

すると、右のような画面が表示されるはずである。
もちろん Firefox や Internet Explorer などの WWW ブラウザでもこの HelloTest.html を見ることができる。



2.1.3 アプレットと HTML

HelloTest.html の中でアプレットの実行に直接関係があるのは、

```
<applet code="Hello.class" width="150" height="50"> </applet>
```

の部分である。code=のあとに読み込みたいアプレットの名前（拡張子.class を付ける）を書く、width と height はアプレットを表示するために確保する領域の幅と高さである。アプレットに対応していないブラウザでは、<applet 〜>〜</applet>の間の HTML（代替テキスト）を表示する。

なお、HTML の最近の規格では、applet タグは非推奨（deprecated）とされ、次のように object タグを用いることが推奨されている。

```
<object codetype="application/java" classid="java:Hello.class"
        width="150" height="50">
</object>
```

しかし、当面はこの新形式をサポートしていないブラウザが多いので、このプリントでは旧来の applet タグを用いて説明する。

> **参考:** HTML ファイルを作成するのが面倒な時、上の Hello.java のように、Java のソースファイル中に applet タグをコメントとして（通常 import 文の後に）挿入しておくと、次のコマンドでアプレットを実行することができる。

```
> appletviewer Hello.java
```

注意: Web ブラウザ上で実行されるアプレットに対して、サーブレット（Servlet）は、Web サーバ側で実行され、HTML などを動的に生成する Java プログラムである。

2.2　Hello アプレット

これで、Java のアプレットを一つ作成し実行することができた。続いて、プログラム（Hello.java）の意味を説明する。

最初の 2 行の import 文は、javax.swing と java.awt という二つの部品群（パッケージ, package）をプログラム中で使用することを宣言している。*はこのパッケージの全てのクラスを使用する可能性があることを示している。典型的な Applet の場合、この 2 行の import 文を用いることが多い。（以降のプリント中の例では自明な場合、この 2 行は省略する。）

詳細: パッケージは OS のディレクトリやフォルダがファイルを階層的に整理するのと 同じように、クラスを階層的に管理する仕組みである。JApplet クラスの正式名称は、パッケージの名前を含めた javax.swing.JAppplet なのであるが、これを単に JApplet という名前で参照できるようにするのに

```
import javax.swing.JApplet;
```

という import 文を使う。javax.swing というパッケージに属するクラスすべてをパッケージ名なしで参照できるようにするのが、

```
import javax.swing.*;
```

という import 文である。

自作のクラスを他のクラスから利用する場合は適切なパッケージに配置するべきである。（自作のクラスをパッケージの中に入れるために package 文というものを使うが、本講義では説明を割愛する。）アプレットやサーブレットの場合は、他のクラスから利用するわけではないので、パッケージなしでも良いだろう。（正確に言うと package 文がない時は、そのファイルで定義されるクラスは無名パッケージというパッケージに属することになる。）

Java の既成のクラスを利用するためには、そのクラスが属するパッケージを調べて、それに応じた import 文を挿入する必要がある。（もしくは、クラスをパッケージ名を含めたフルネームで参照する。）

次の public class Hello extends JApplet は、JApplet という _____を _____して（つまり、ほんの少し書き換えて）、新しいクラス Hello を作ることを宣言している。（この時、Hello クラスは JApplet クラスのサブクラス、逆に JApplet クラスは Hello クラスのスーパークラスと言う。）

JApplet クラスは、アプレットを作成する時の基本となるクラスで、アプレットとして振舞うための基本的なメソッドが定義されている。すべてのアプレットはこのクラスを継承して定義する。このため、必要な部分だけを再定義すれば済む。

行の最初の public はこのクラスの定義を外部に公開することを示している。逆に、公開しない場合は、private というキーワードを使う。（このプリントでは、はじめのうちは、public なクラスしか定義しない。）

Hello クラスは JApplet クラスの paint という名前のメソッドを上書き（＿＿＿＿＿＿＿＿＿＿）している。クラスを継承する時は元のクラス（スーパークラス）のメソッドを上書きすることもできるし、新しいメソッドやインスタンス変数を加えることもできる。paint クラスの定義の前の行の @Override は JDK5.0 から導入された＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿というもので、スーパークラスのメソッドをオーバーライドすることを明示的に示すものである。これにより、スペリングミスなどによるつまらない（しかし発見しにくい）バグを減らすことができる。

> **参考: クラス名に使える文字の種類**　Java では、クラス名に次の文字が使える（変数名、メソッド名なども同じ。）このうち数字は先頭に用いることはできない。
>
> アンダースコア（“_”）, ドル記号（“$”）, アルファベット（“A”〜“Z”, “a”〜“z”）, 数字（“0”〜 “9”）, かな・漢字など（Unicode 表 0xc0 以上の文字）
>
> Java は C 言語と同じようにアルファベットの大文字と小文字は、＿＿＿＿＿＿。その他にクラス名は大文字から始める、などのいくつかの決まりとまでは言えない習慣がある。public や void, for, if のように Java にとって特別な意味がある単語（＿＿＿＿＿＿＿＿）はクラス名などには使えない。
>
> ドル記号とかな・漢字を用いることができるところが C や C++との違いである。

main はどこに行った?　このプログラムには main 関数がない。アプレットは Web ブラウザの中で動作させることのできるプログラムの一部にすぎず、従来の意味でのプログラムではない。だからアプレットの main 関数にあたるものは Web ブラウザの main 関数であると言える。

2.3 JApplet クラス、Graphics クラスのメソッド

アプレットにはいくつかのメソッドがあり、必要に応じてブラウザによって呼び出される。例えば、paint メソッドは、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿に呼び出される。

このようなイベントが起こった時に呼び出されるメソッドは、主なものだけでも次のようなものがある。

init メソッド　ブラウザが＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿に 1 回だけ呼ばれる。

start メソッド　ブラウザが＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿に呼ばれる[4]。

stop メソッド　＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿に呼ばれる[5]。

paint メソッドは

```
public void paint(Graphics g)
```

[4] init がよばれた後、あるいは他のページからアプレットのあるページに戻ってきた時など

[5] 他のページにジャンプする時など

という部分から、＿＿＿＿＿＿＿＿のオブジェクトを引数として受け取ること、戻り値はないことがわかる。`public` というキーワードがついていることと、`class` の定義の中に埋め込まれていることを除けば、C 言語の関数定義の方法と同じ書き方である。

`Graphics` クラスはいわば絵筆に対応するデータ型で、“絵の具の色” や “太さ” にあたるデータを構成要素（インスタンス変数）として持っている。このクラスのオブジェクトを使って画面上に文字や絵をかくことができる。`Hello` クラスでは、この `Graphics` クラスの `drawString` というメソッドを使って、“Hello World!” という文字を書いている。後ろの 50 と 25 は、表示する位置である。

`void drawString(String str, int x, int y)`　座標 (x,y) に文字列 `str` を描画する。

2.4　メソッド呼び出し

このように Java ではオブジェクトのメソッドを呼び出すために、

オブジェクト ＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

という形を用いる。また、インスタンス変数（フィールド）をアクセスする時は、

オブジェクト ＿＿＿＿＿＿＿＿

という書き方を用いる。前述したようにオブジェクトはいくつかのデータをまとめて一つの部品として扱えるようにした物であり、`.`（ドット）演算子は、＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿演算子である。つまり、`g.drawString( ... )` は、`g` という `Graphics` クラスのオブジェクトから `drawString` というメソッドを取り出して引数を渡す式である。（Java のメソッドは必ずクラスの中で定義されている。そのため、同じオブジェクトのメソッドを呼出すなど特別な場合をのぞき、Java のメソッド呼出しには、このドットを使った記法が必要である。メソッドのドキュメントにはこの部分は明示されないので注意が必要である。）

> **参考:** `.` 演算子の前に書く値も、メソッド名の後の括弧の間に `,` 区切りで書く値も、どちらもメソッドに渡されるデータという意味では違いはないが、上述のようにイメージが異なる。`.` 演算子の前にあるのは “主語” で、括弧の間にある通常の引数は “目的語” のようなイメージである。
>
> メソッドはクラスの中に定義されているので、同じ名前のメソッドが複数のクラスで定義されていて、同じ名前のメソッドでもクラスが異なれば実装が異なることがある。`.` 演算子の前のオブジェクトが、どのメソッドの実装を呼び出すかを決定する。
>
> この点は**動的束縛**を説明するときに、より詳しく説明する。

問 2.4.1

1. *Hello.java* の *"Hello World!"* の部分を書き換えて、他の文字列を表示させよ。
2. *Hello.java* の *50, 25* の部分を書き換えて表示する位置を変更せよ。

2.5 アプレットのパラメータ

次の例では _____というタグを使って、アプレットに HTML 側からパラメータ（引数）を渡している。このタグは <applet ... >と</applet>の間にはさむ。param タグに必要なものはパラメータの名前（_____）とその値（_____）である。プログラム中でパラメータの値を得るためには JApplet クラスの _____ というメソッドを用いる。このメソッドは、パラメータの名前を引数として受け取って、そのパラメータの値を返す。

例題 2.5.1

ファイル GreetingTest.html

```
<html>
<head></head>
<body>
<applet code="Greeting.class" width="150" height="50">
  <param name="String" value="Bon Jour!">
</applet>
</body>
</html>
```

ファイル Greeting.java（その 1）

```
import javax.swing.*;
import java.awt.*;

public class Greeting extends JApplet {
  @Override
  public void paint(Graphics g) {
    String theArg;
    theArg = getParameter("String");
    g.drawString(theArg, 50, 25);
  }
}
```

上のアプレットを実行すれば、変数 *theArg* に、パラメータ *String* の値 *Bon Jour!* が入るので、画面に *"Bon Jour!"* と表示される。表示される文字列を *"Guten Tag!"* に変更したいときは、*GreetingTest.html* を書き換えれば済み、*Greeting.java* を再コンパイルする必要はない。

2.6 変数の宣言

Greeting.java（その 1）の theArg の例でわかるように変数の宣言は C と同様、

_____ 変数名 _

の形式で行なう。型名は int, double などのプリミティブ型か、クラス名である。ただし、C と違って、使用する前に宣言すれば必ずしも関数定義の最初に宣言する必要はない。変数への代入も C と同様 _____を使う。

2.7　インスタンス変数の宣言

Greeting.java（その 1）では画面を描き直すたびに getParameter メソッドが呼ばれ、効率は良くない。getParameter メソッドは何度呼ばれても引数が同じならば値は変わらないので、init メソッドで一度だけ getParameter メソッドが呼ばれるように書き直す。

例題 2.7.1

ファイル Greeting.java（その 2）

```
import javax.swing.*;
import java.awt.*;

public class Greeting extends JApplet {
  String theArg;

  @Override
  public void init() {
    // init はアプレットの初期化の時に一度だけ呼ばれる。
    theArg = getParameter("String");
  }

  @Override
  public void paint(Graphics g) {
    g.drawString(theArg, 50, 25);
  }
}
```

このプログラムでは、theArg をインスタンス変数として宣言している。インスタンス変数の宣言は、クラス定義の中に、メソッドの定義と同じレベルに（メソッドの定義の外に）並べて書く。インスタンス変数はそのクラス中のすべてのメソッドから参照することができる。（ある意味で C 言語の大域変数と似ている。）

前述したようにクラスはオブジェクトの雛型である。Greeting クラスに theArg というインスタンス変数を宣言したということは、Greeting クラスのインスタンスが作られる時に、JApplet クラスが持っているすべての構成要素の他に、theArg という名前の、String 型の構成要素ができるということである。

init メソッドの中でこの theArg に値を代入し、paint メソッドの中で参照している。このようにメソッドの中で _____は、ピリオドを使った記法は必要ない。（getParameter メソッドも JApplet クラスのメソッドであるため、paint や init メソッドの中ではピリオドを使った記法は必要ない。）

インスタンス変数はオブジェクトが存在している間は値を保持している。これに対して、メソッドの中で宣言された変数（例えば、Greeting.java（その 1）の theArg）の寿命は paint メソッドの呼出しの間だけである。2 度め以降の呼び出しでも以前の値は保持していない。

Java のコメント　Java のコメントには C と同じ形式の _____と _____の間、という形の他にも、上の例のように _____から行末まで、という形式も使える。（C++と同じ。最近の C の仕様（C99）でも //〜形式のコメントが使えるようになっている。）

問 2.7.2 *(JDKDIR)*[6]*/demo/Blink/*フォルダ、*(JDKDIR)/demo/NervousText/*の内容を適当なところにコピーして、*HTML* ファイルのパラメータを変更して実行せよ。

問 2.7.3 *(JDKDIR)/demo* フォルダなどから、その他の適当なできあいのアプレットを探してきて、パラメータを変えて実行せよ。

> **参考: エラーメッセージのリダイレクト**　Java のソースファイルをコンパイルすると、エラーメッセージが大量に出力されて、最初の方のメッセージが見えないという事態が起こることがある。この場合は、エラーメッセージを別のファイル（ログファイル）に書き込んで（リダイレクトという）、あとでログファイルをメモ帳などで見るようにすると良い。
>
> リダイレクトは次のような方法で行なう。
>
> ```
> > javac ソースファイル 2> ログファイル
> > javac -J-Djavac.pipe.output=true ソースファイル > ログファイル
> ```
>
> 上段が Windows NT/2000/XP/Vista の場合、下段が Windows 95/98/Me の場合である。

2.8 Java のグラフィクス（AWT）— 色とフォント

Hello.java では、Graphics クラスの drawString メソッドを使って、画面に文字を表示したが、ここではこのクラスの他の描画メソッドを紹介する。

Graphics オブジェクトの色とフォントは次のメソッドで変更することができる。

void setColor(Color c)	_____する。
void setFont(Font f)	_____する。

例題 2.8.1

ファイル ColorTest.java

```
import javax.swing.*;
import java.awt.*;

public class ColorTest extends JApplet {
  @Override
  public void paint(Graphics g) {
    g.setColor(Color.BLUE);
    g.drawString("Good Morning!", 20, 25);
    g.setColor(Color.ORANGE);
    g.setFont(new Font("TimesRoman", Font.BOLD, 14));
    g.drawString("Good Afternoon!", 20, 50);
    g.setColor(Color.RED);
    g.setFont(new Font("TimesRoman", Font.ITALIC, 14));
    g.drawString("Good Evening!", 20, 75);
  }
}
```

[6] *(JDKDIR)* は JDK をインストールしたディレクトリのことを指すことにする。これは JDK のバージョンにより異なる。

HTML ファイルの方ではアプレットの領域の高さを増やしておく（*height="100"*くらい）必要がある。実行すると左の図のようになる。
色を指定するためには、*Color* クラスのインスタンスを使う。上のプログラムのように、*Color.BLUE*, *Color.RED* など定数（正確にはクラス変数）として用意されているインスタンス[a]を用いる方法の他にも *RGB* 値を直接指定して新しい *Color* クラスのインスタンスを生成する方法もある。

[a] BLUE, RED, ORANGE の他に BLACK, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, PINK, WHITE, YELLOW が用意されている。

Tips: Java のグラフィックス関数では標準ではアンチエイリアシングを行わないので、斜めの線の輪郭がギザギザに見えて美しくない。アンチエイリアシングをするには、描画の前に

```
((Graphics2D)g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                                 RenderingHints.VALUE_ANTIALIAS_ON);
```

という呼出しをしておけば良い。setRenderingHint は Graphics のサブクラスの Graphics2D クラスのメソッドである。JApplet の paint メソッドに渡される引数は、実際には Graphics2D であるが、普段は Graphics クラスとして扱われるので、Graphics2D クラスのメソッドを利用するためにキャスト（型変換）を行っている。

2.9　クラス変数とクラスメソッド

クラス変数は _____ であり、クラスメソッドは _____ である。どちらも、クラスによって決まるので、.（ドット）演算子の左にクラス名を書くことによってアクセスできる。以前に登場した System.out も System というクラスの out という名前のクラス変数である。

クラスメソッド・クラス変数のことを、それぞれスタティックメソッド・スタティック変数と呼ぶこともある。これは、クラス変数やクラスメソッドを定義する時に _____ という修飾子をつけるためである。API 仕様のドキュメントにも static と付記される。例えば、Color クラスのドキュメントの中では、

```
static Color BLACK
```

Math クラスのドキュメントの中では、

```
static double cos(double a)
```

のように説明されている。これはそれぞれ使用するときには、_____, _____ のようにクラス名+.+メソッド名の形に書かなければいけないということを示している。

また、Java アプリケーションで必ず定義する main メソッドも、スタティックメソッドでなければならない。

参考: Java 5.0 以降では static import という仕組みを利用することで、クラス変数・メソッドの前のクラス名を省略することができるようになった。例えば、プログラムの先頭に、

```
import static java.lang.Math.cos;  // cos 関数だけの場合、
// または
import static java.lang.Math.*;    // Math クラスの全てのスタティックメンバ
```

と書くと、単に cos(0.1) のように書くことができる。

2.10 インスタンスの生成

一般に、あるクラスのインスタンスを生成するには、____という演算子を使う。new の次に _____ _____（constructor）という、クラスと同じ名前のメソッドを呼び出す式を書く。コンストラクタに必要な引数は各クラスにより異なるので API ドキュメントを調べる必要がある。また、ひとつのクラスが引数の型が異なる複数のコンストラクタを持つ場合もある。

Color クラスの場合、コンストラクタは 3 つの int 型の引数をとる。それぞれ 0 から 255 の範囲で赤（R）・緑（G）・青（B）の強さを表す。つまり、new Color(255,0,0)は純粋な赤を表す Color オブジェクトになる。g.setColor(Color.RED);は g.setColor(_____);でも同じ意味になる。

Font クラスのコンストラクタは、フォントの種類を表す文字列（”Serif”の他、”Monospaced”, ”SansSerif”, ”Dialog”, ”DialogInput”のどれか）、スタイル （Font.BOLD （太字体）, Font.ITALIC （斜字体）, Font.PLAIN（通常の字体）の 3 つの定数のどれか）、サイズを表す整数、の 3 つの引数をとる。new Font("Serif", Font.BOLD, 16) は、セリフ体の太字体の 16 ドットのサイズのフォントである。

問 2.10.1 例題を改造して、いろいろな色・フォント・文字列を組み合わせを試せ。

2.11 図形の描画

Graphics クラスは、直線、長方形、多角形、楕円、円などさまざまな図形を描画するためのメソッドを持つ。

`void drawLine(int x1, int y1, int x2, int y2)`
(x1, y1) から (x2, y2) まで直線を引く。
`void drawRect(int x, int, y, int w, int h)`
左上の点が (x, y) で幅 w, 高さ h の長方形を描く。
`void clearRect(int x, int, y, int w, int h)`
左上の点が (x, y) で幅 w, 高さ h の長方形の領域をバックグラウンドの色で塗りつぶす。
`void drawOval(int x, int y, int w, int h)`
左上の点が (x, y) で幅 w, 高さ h の長方形に内接する楕円を描く。
`void drawPolygon(int[] xs, int[] ys, int n)`
(x[0], y[0])〜(x[n-1], y[n-1]) の各点を結んでできる多角形を描く。
`void fillRect(int x, int, y, int w, int h)`
左上の点が (x, y) で幅 w, 高さ h の長方形を描き塗りつぶす。

一般に、draw〜という名前のメソッドは内部を塗りつぶさず、fill〜は内部を塗りつぶす。

注意: Java のグラフィックスの座標系は左上の点が原点で、x 軸は通常と同じく右に向かって増えていくが、y 軸は数学で使われる座標軸と違って、下に向かって増えていく。単位はピクセル（画素）である。

重要: Java のクラスやメソッドは Java API 仕様というドキュメントにまとめられている。Java 6 の場合、http://java.sun.com/javase/ja/6/docs/ja/api/index.html の左下のフレームからクラスを選択することによって、そのクラスに定義されているメソッド・フィールド・コンストラクタなどの仕様が上の Graphics クラスのメソッドの説明のような形式で示されている。今後は必要に応じてこのドキュメントを調べると良い。

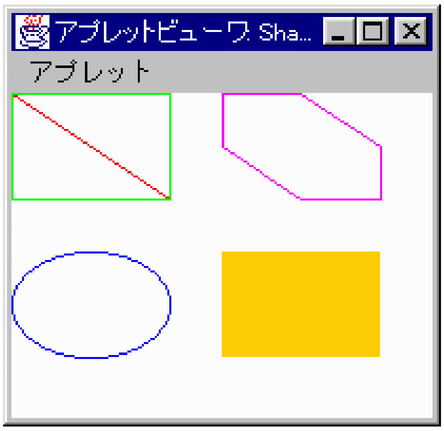
例題 2.11.1

ファイル *ShapeTest.java*

```
import javax.swing.*;
import java.awt.*;
import static java.awt.Color.*;

public class ShapeTest extends JApplet {
  public static int[] xs = { 100, 137, 175, 175, 137, 100};
  public static int[] ys = {   0,   0,  25,  50,  50,  25};

  @Override
  public void paint(Graphics g) {
    g.setColor(RED);
    g.drawLine(0, 0, 75, 50);
    g.setColor(GREEN);
    g.drawRect(0, 0, 75, 50);
    g.setColor(BLUE);
    g.drawOval(0, 75, 75, 50);
    g.setColor(MAGENTA);
    g.drawPolygon(xs, ys, 6);
    g.setColor(ORANGE);
    g.fillRect(100, 75, 75, 50);
  }
}
```



drawLine, *drawRect* などはすべて *Graphics* クラスのメソッドであるので、________________________ __________ことに注意する。

このクラスでは、2 つの変数 *xs*, *ys* をクラス変数として宣言している。これらのクラス変数は、メソッド *paint* の中で *drawPolygon* の引数として用いられている。

このプログラムの実行結果は左の図のようになる。

2.12 配列の宣言

```
public int[] xs = {100, 137, 175, 175, 137, 100};
```

は、C 言語では

```
int xs[] = {100, 137, 175, 175, 137, 100};
```

と書くべきところだが、Java ではどちらの書き方（[] の位置に注意）も可能である。[] は型表現の一部であるということを強調するため、Java では前者の書き方をすることが望ましい。

問 2.12.1 *ShapeTest.java* の数値・色をなどをいろいろ変えて試せ。

問 2.12.2 その他の *Graphics* クラスのメソッド:

void draw3DRect(int x, int y, int w, int h, boolean raised)
void drawArc(int x, int y, int w, int h, int angle1, int angle2)
void drawRoundRect(int x, int y, int w, int h, int rx, int ry)
void fillOval(int x, int y, int w, int h)
void fillPolygon(int[] xs, int[] ys, int n)
void fill3DRect(int x, int y, int w, int h, boolean raised)
void fillArc(int x, int y, int w, int h, int angle1, int angle2)
void fillRoundRect(int x, int y, int w, int h, int rx, int ry)

はどのような図形を描くか、試せ。

問 2.12.3 *String*（正式には *java.lang.String*）クラスのドキュメントで、次のような機能を持つメソッドの使い方を調べよ。

1. 指定された文字が最初に出現する位置を返す。
2. 指定された文字が最後に出現する位置を返す。
3. 文字列の *m* 文字目から *n* 文字目までの部分文字列を取り出す。

実際にそれらを使用して、次のようなプログラムを作成せよ。（いずれも最初の文字は *0* 文字目と数える。）

1. 文字列 *"The quick brown fox jumps over the lazy dog."*の中で最初に 文字 *a* が現れる位置を表示する。
2. 同じ文字列 *"The quick brown ～."*の中で最後に 文字 *e* が現れる位置を表示する。
3. 同じ文字列 *"The quick brown ～."*の *11* 文字目から *20* 文字目を取り出す。

キーワード JDK, class、javac, java、main メソッド、アプレット、import、appletviewer、JApplet クラス、 継承、extends、オーバーライド、paint メソッド、init メソッド、start メソッド、stop メソッド、Graphics クラス、drawString メソッド、applet タグ、param タグ、インスタンス変数、クラス変数、クラスメソッド、new 演算子、 コンストラクタ、Java API 仕様、配列

メモ: