

## 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（現 Oracle 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、`javac` という Java コンパイラーと `java` という中間コード実行プログラム（JVM エミュレーター）である。

統合開発環境（IDE）としては、Sun Microsystems 社（現 Oracle 社）の NetBeans、IBM<sup>1</sup> によって開発された Eclipse<sup>2</sup>、JetBrains 社の IntelliJ IDEA などがある。IDE は、エディター・コンパイラー・デバッガー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 コンソールアプリケーション

例題 2.1.1 まず、通常のアプリケーション（つまり後述の GUI を用いない）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ー（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、`javac` というコマンドを用いる。

```
> javac Hello0.java
```

<sup>1</sup>現在は IBM とは独立した組織 Eclipse Foundation で開発されている。

<sup>2</sup><http://www.eclipse.org/>

コンパイルが成功すれば、同じディレクトリーに Hello0.class というファイルができています。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 `java` で実行する。

```
> java Hello0
```

(.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) へのコンパイラー
java   — JVM のエミュレーター
```

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
```

**Q 2.1.2** Foo.java という (無名パッケージの) Java のソースファイルを中間言語にコンパイルするためのコマンドを書け。

答: \_\_\_\_\_

**Q 2.1.3** Bar.class という main メソッドを含む (無名パッケージの) クラスの Java の中間言語ファイルを実行するためのコマンドを書け。

答: \_\_\_\_\_

Hello0.java の意味を簡単に説明する。

`public class Hello0` は Hello0 という \_\_\_\_\_ (空欄 2.1.1) 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、とりあえずこの形のまま使えば良い。) 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 \_\_\_\_\_ (空欄 2.1.2)。<sup>3</sup> この例の場合はどちらも Hello0 でなければならない。この後の開ブレース ( { ) 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。

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。

<sup>3</sup>public でないクラス名に対しては、この規則は強制されない。



ていないこと、このテキストで扱う範囲の基本的な考え方は Swing と大差ないことから、ここでは Swing を選択する。

次のようなファイル (Hello.java) をエディター (秀丸、メモ帳、Emacs など) で作成する。

ファイル Hello.java

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

public class Hello extends JPanel {
    public Hello() {
        setPreferredSize(new Dimension(250, 50));
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("HELLO_WORLD!", 50, 25);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("こんにちは"); // タイトル部分
            frame.add(new Hello());
            frame.pack();
            frame.setVisible(true);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        });
    }
}
```

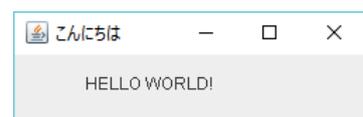
Hello.java を JVM のコードにコンパイルするためには、やはり `javac` コマンドを用いる。

```
> javac Hello.java
```

コンパイルが成功すれば、同じディレクトリーに Hello.class というファイルができています。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、Hello を `java` で実行する。

```
> java Hello
```

すると、右のような画面が表示されるはずである。



Java のコメント Java のコメントには C と同じ形式の \_\_\_\_\_ (空欄 2.1.4) と \_\_\_\_\_ (空欄 2.1.5) の間、という形の他にも、 \_\_\_\_\_ (空欄 2.1.6) 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る。)

## 2.2 Hello クラス

これで、Java の GUI アプリケーション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、このパッケージのすべ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 GUI アプリケーションの場合、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 のディレクトリーやフォルダー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。JPanel クラスの正式名称は、パッケージの名前を含めた javax.swing.JPanel なのであるが、これを単に JPanel という名前で参照できるようにするのに

```
import javax.swing.JPanel;
```

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 (後述) というものを使う。このテキストで紹介するような main メソッドを持つクラスや、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 文がないときは、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。

注意: Java のソース (.java) ファイルやクラス (.class) ファイルは、そのパッケージに対応するディレクトリーに配置することになっている。このテキストでは当面の間、無名パッケージのクラスのみを作成するので、カレントディレクトリーの直下にソース (.java) ファイルもクラス (.class) ファイルも置かれていると想定する。他の場所に置くと、コンパイルや実行ができないことがある。



## 2.3 Graphics クラスのメソッド

JPanel にはいくつかのメソッドがあり、必要に応じて呼び出される。例えば、`paintComponent` メソッドは、画面を \_\_\_\_\_ (空欄 2.3.1) ときに呼び出される。

`paintComponent` メソッドは

```
public void paintComponent(Graphics g)
```

という部分から、`Graphics` 型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。

`paintComponent` メソッドは、その中で `super.paintComponent(g)` を呼び出している。`super.~` はスーパークラスで定義されているメソッドを呼び出すための書き方である。`super.paintComponent(g)` は背景を再描画する働きがある。

`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 “字の形” にあたるデータを構成要素 (フィールド) 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!” という文字列を書いている。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.メソッド名(引数<sub>1</sub>, ..., 引数<sub>n</sub>)

という形を用いる。また、フィールド (インスタンス変数) をアクセスするときは、

オブジェクト.フィールド名

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、オブジェクトの中から \_\_\_\_\_ (空欄 2.4.1) 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: . (ドット) 演算子の前に書く値も、メソッド名の後の括弧の間に , (コンマ) 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。

**Q 2.4.1** g という名前の変数が Graphics 型のオブジェクトのとき、座標 (12, 34) に “Thank You!” という文字列を表示するメソッド呼出しの式を書け。

答: \_\_\_\_\_

#### 問 2.4.2

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 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) \_\_\_\_\_ (空欄 2.5.1) する。

void setFont(Font f) \_\_\_\_\_ (空欄 2.5.2) する。

#### 例題 2.5.1

ファイル ColorTest.java

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

public class ColorTest extends JPanel {
    public ColorTest() {
        setPreferredSize(new Dimension(150, 100));
    }

    @Override
    public void paintComponent(Graphics g) {
```

```

super.paintComponent(g);
String msg = "Hello, World!";
g.setColor(Color.BLUE);
g.setFont(new Font(Font.SERIF, Font.PLAIN, 14));
g.drawString(msg, 20, 25);
g.setColor(Color.ORANGE);
g.setFont(new Font(Font.SERIF, Font.BOLD, 14));
g.drawString(msg, 20, 50);
g.setColor(Color.RED);
g.setFont(new Font(Font.SERIF, Font.ITALIC, 14));
g.drawString(msg, 20, 75);
}

... // main メソッドの定義は割愛
}

```



コンストラクターでは、領域の高さを増やしておく(100くらい)必要がある。実行すると左の図のようになる。

変数の宣言 変数の宣言はCと同様、

\_\_\_\_\_ (空欄 2.5.3) 変数名 \_\_\_\_\_ (空欄 2.5.4)

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 \_\_\_\_\_ (空欄 2.5.5) を使う。

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<sup>4</sup>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。

**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。JPanel の

<sup>4</sup>BLUE, RED, ORANGE の他に BLACK, CYAN, DARKGRAY, GRAY, GREEN, LIGHTGRAY, MAGENTA, PINK, WHITE, YELLOW が用意されている。

paintComponent 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。

## 2.6 クラスフィールドとクラスメソッド

\_\_\_\_\_ (空欄 2.6.1) (クラス変数) はクラスに属するオブジェクトから共通にアクセスされる変数であり、\_\_\_\_\_ (空欄 2.6.2) は、通常のフィールドにアクセスせずクラスフィールドだけにアクセスするメソッドである。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 (正確に言うところでは java.lang.System) というクラスの out という名前のクラスフィールドである。

クラスメソッド・クラスフィールドのことを、それぞれ **スタティックメソッド・スタティックフィールド** と呼ぶこともある。これは、クラスフィールドやクラスメソッドを定義するときに \_\_\_\_\_ (空欄 2.6.3) 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 (java.lang.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、Color.BLACK、Math.cos(0.1) のようにクラス名・メソッド名 (またはフィールド名) の形に書かなければいけないということを示している。

なお、Java の static は、C 言語での static の使い方とはあまり関連がない。

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、クラスフィールド・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 クラスのすべてのクラスフィールド・クラスメソッド
```

と書くと、単に cos(0.1) のように書くことができる。

**Q 2.6.1** g という名前の変数が Graphics 型のオブジェクトのとき、以後の描画を緑色にするメソッド呼出しの式を書け。ただし、import java.awt.\*; はしているが import static java.awt.Color.\*; はしていないと仮定する。

答: \_\_\_\_\_

Q 2.6.2 円周率  $\pi$  を表す定数は `java.lang.Math` クラスの中で

```
public static final double PI = 3.141592653589793;
```

と宣言されている。( `final` 修飾子は後述するが、この問に関しては無視してよい。)

( `static import` を使わないとき ) 円周率を表す Java の式を書け。

答: \_\_\_\_\_

メソッド呼出しのまとめ

結局、Java のメソッド呼出しは、次の3通りの形式が存在することになる。

1. オブジェクト `.` メソッド名 (引数の並び)  
例: `g.drawString("Hello", 20, 75)`
2. クラス名 `.` メソッド名 (引数の並び)  
例: `Math.cos(0)`
3. メソッド名 (引数の並び) ... ドットを使わない形  
例: `setPreferredSize(new Dimension(150, 100))`

このうち、1 が、もっとも普通のメソッド呼出しである。通常、API ドキュメントでは、次のようにメソッド名と引数の型だけが紹介される。

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

しかし、使うときには、`g.drawString("HELLO WORLD!", 50, 25)` のように、オブジェクトを表す式とドットを前につける必要がある。

また、2 はクラスメソッドの呼出しである。API ドキュメントでは、次のように `static` というキーワードを伴って紹介される。

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

この場合は、`Math.cos(Math.PI)` のように、クラス名 (この場合は `Math`) とドットを前につける必要がある。

ここまではいずれにしても「`.`」(ドット)とその前に何かが必要である。ドットを使わず、3 のように書くのは、以下の2つの場合である。

- `static import` を使っている場合、
- 同じオブジェクトのメソッドを呼び出す場合、

例えば、`import static java.lang.Math.*;` のような宣言があれば、単に `cos(PI)` のように書くことができる。

また、`setPreferredSize` メソッドは、`JPanel` クラス (正確には、`JPanel` のスーパークラスの `JComponent` クラス) に用意されているメソッドなので、`JPanel` を継承するクラスの (非 `static` な) メソッドからは、単に `setPreferredSize(new Dimension(100, 100))` のように呼出すことができる。



(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 8 の場合、<http://docs.oracle.com/javase/8/docs/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・コンストラクターなどの仕様が上の 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 2.8.1

ファイル ShapeTest.java

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

public class ShapeTest extends JPanel {
    public ShapeTest() {
        setPreferredSize(new Dimension(200, 150));
    }

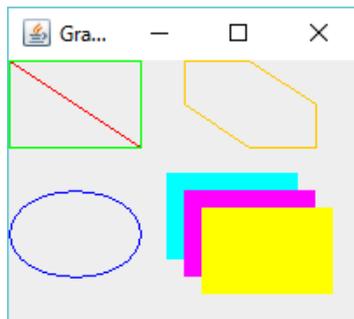
    @Override
    public void paintComponent(Graphics g) {
        int[] xs = { 100, 137, 175, 175, 137, 100};
        int[] ys = { 0, 0, 25, 50, 50, 25};

        super.paintComponent(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(ORANGE);
        g.drawPolygon(xs, ys, 6);
        g.setColor(CYAN);
        g.fillRect(90, 65, 75, 50);
    }
}
```

```

g.setColor(MAGENTA);
g.fillRect(100, 75, 75, 50);
g.setColor(YELLOW);
g.fillRect(110, 85, 75, 50);
}
}

```



drawLine, drawRectなどはすべてGraphicsクラスのメソッドであるので、Graphicsクラスのオブジェクトgのメソッドを呼び出すため、`g.drawLine(x1, y1, x2, y2)` (空欄 2.8.1) という形式で使用されていることに注意する。

このプログラムでは、2つの変数xs, ysを宣言している。これらの変数は、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。

#### 配列の宣言 配列の宣言の

```
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では前者の書き方が好まれる。

問 2.8.2 ShapeTest.java の数値・色などをいろいろ変えて試せ。

問 2.8.3 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、GUIアプリケーションを作成して試せ。

**boolean 型** `boolean` 型は真偽値 ( \_\_\_\_\_ (空欄 2.8.2) か \_\_\_\_\_ (空欄 2.8.3) の 2 つの値 ) を取り得る型である。

問 2.8.4 指定された文字が最初に出現する位置 (最初から数えて何文字めか) を返すメソッド `indexOf` の使用例にならい、`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:

1. 指定された文字が最後に出現する位置 (最初から数えて何文字めか) を返す。
2. 文字列の `m` 文字目から `n-1` 文字目までの部分文字列を取り出す。

の使い方を調べて、実際にそれらを使用して、次の要件を満たす一つの Java コンソールアプリケーションを作成せよ。(いずれも最初の文字は 0 文字目と数える。空白も 1 文字と数える。)

1. 同じ文字列 `msg` の中で最後に文字 'e' が現れる位置を表示する。(33 文字目のはず)
2. 同じ文字列 `msg` の 11 文字目から 20 文字目を取り出して表示する。("rown fox j"になるはず)

なお、文字リテラル (定数) は C 言語と同様、一重引用符に囲んで 'a' のように表す。

解答のテンプレート

ファイル `StringExample.java`

```
public class StringExample {
    public static void main(String[] args) {
        String msg = "The_quick_brown_fox_jumps_over_the_lazy_dog.";
        System.out.printf("文字 'a' は %d 文字目に現れます。%n",
                          msg.indexOf('a'));
        ...
    }
}
```

## 2.9 キーボードからの入力

Java でキーボードからの入力を扱うときは `java.util.Scanner` というクラスを用いる。次の例は、このクラスのオブジェクトの典型的な使用例である。

ファイル `ScannerExample.java`

```
import java.util.Scanner;

public class ScannerExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
int i = sc.nextInt();
System.out.printf("%dの2乗は%dです。 %n", i, i*i);
}
}
```

このクラスの `nextInt` というメソッドは、次の単語を `int` としてスキャンする。他に同じクラスのメソッドとして、次の単語を `double` としてスキャンする `nextDouble` や、次の単語（単語の区切りは空白またはタブ文字）をスキャンする `next`、次の改行までスキャンする `nextLine` などのメソッドがある。

問 2.9.1 `java.util.Scanner` クラスの `nextDouble`, `next`, `nextLine` メソッドを使用したプログラムを作成せよ。

キーワード `JDK`、`class`、`javac`、`java`、`main` メソッド、`import`、`JFrame` クラス、`JPanel` クラス、継承、`extends`、オーバーライド、`paintComponent` メソッド、`Graphics` クラス、`drawString` メソッド、フィールド、クラスフィールド、クラスメソッド、`new` 演算子、コンストラクター、Java API 仕様、配列、`boolean` 型、