

プログラミング I

Java Appletプログラミング

文教大学 情報学部 経営情報学科
堀田 敬介

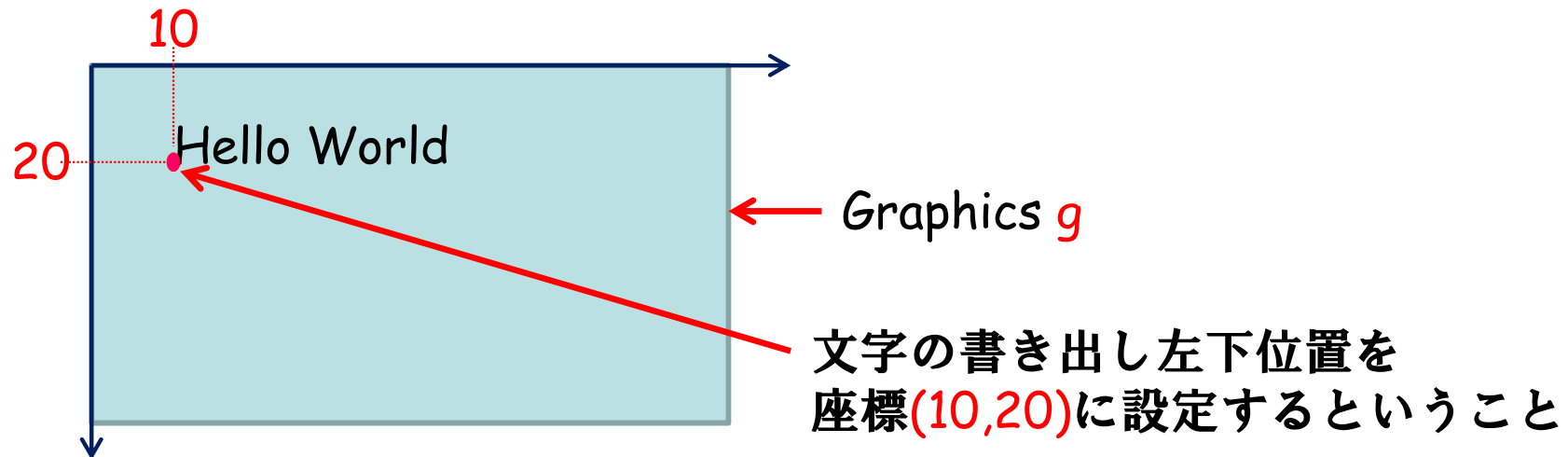
アプレット Applet

```
import java.applet.*;  
import java.awt.*;
```

```
public class クラス名 extends Applet {
```

```
    public void paint(Graphics g) {    // アプレット 描画  
        g.drawString("Hello World", 10, 20);  
    }
```

```
}
```



Eclipse (Ver3.7) で実行する場合, 実行前に

- ①メニューの「実行」-「実行構成」を選び,
- ②「パラメータ」の「縦」と「横」のサイズを変更する

(しなくても, ウィンドウ表示後にサイズ変更して再実行(再描画)できるので問題はない)

アプレット: 色の設定法

```
import java.applet.*;
import java.awt.*;
```

```
public class ... extends Applet {
    Color col;           // カラークラス
    int red, grn, blu;
```

```
    public void paint(Graphics g) {
        g.setColor(Color.red);
```

```
        // カラークラス
        col = new Color(0x00FF99);
        g.setColor(col);
```

```
        red = 255; grn = 0; blu = 125;
        col = new Color(red, grn, blu);
        g.setColor(col);
```

```
    }
```

```
}
```

光の強さ
【16進数】

弱	00	10	...	F0
	01	11	...	F1
	:	:	...	:
	0F	1F	...	FF強

光の強さ
【10進数】

弱	0	16	...	240
	1	17	...	241
	:	:	...	:
	15	31	...	255強

色	指定法	色	指定法
白	<i>Color.white</i>	赤	<i>Color.red</i>
薄灰	<i>Color.lightGray</i>	青	<i>Color.blue</i>
灰	<i>Color.Gray</i>	緑	<i>Color.green</i>
濃灰	<i>Color.darkGray</i>	淡紅	<i>Color.pink</i>
黒	<i>Color.black</i>	橙	<i>Color.orange</i>
		シアン	<i>Color.cyan</i>
		マゼンタ	<i>Color.magenta</i>
		イエロー	<i>Color.yellow</i>

準備色8色 + 濃淡5色

// 色を設定するメソッド(準備色指定)

// 16進数で光3原色をつくる
// 作った色を設定

// 10進数で光3原色をつくる
// 作った色を設定

アプレット:フォントの設定法

```
import java.applet.*;
import java.awt.*;
```

```
public class ... extends Applet {
```

```
    Font ft;           // フォントクラス
```

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

```
        ft = new Font("Elephant", Font.PLAIN, 24); // フォントをつくる
```

```
        g.setFont(ft); // フォント設定メソッド
```

```
    }
```

```
}
```

サイズ



指定できるフォントの種類(例)

指定できる書体

フォント(英字)	フォント(日本語)
Rockwell	MS ゴシック
Broadway	MS 明朝
Times New Roman	HGP行書体
Georgia	HG教科書体
Tekton Pro	null (指定せず)

書体	意味
Font.PLAIN	普通
Font.BOLD	太字
Font.ITALIC	斜体
Font.BOLD Font.ITALIC	太字斜体

アプレット: 基本図形描画

```
import java.applet.*;
import java.awt.*;

public class ... extends Applet {
    public void paint(Graphics g) {
        g.drawLine(10, 10, 100, 30);           // 線分

        g.drawRect(10, 10, 100, 50);           // 矩形
        g.fillRect(10, 10, 100, 50);           // 塗り潰し矩形

        g.drawOval(10, 10, 70, 50);            // 楕円
        g.fillOval(10, 10, 70, 50);            // 塗り潰し楕円

        g.drawRoundRect(10, 10, 100, 50, 5, 5); // 角丸矩形
        g.fillRoundRect(10, 10, 100, 50, 5, 5); // 塗り潰し角丸矩形

        int x[] = {10, 60, 70}, y[] = {30, 30, 60}, pt = 3;
        g.drawPolygon(x, y, pt);               // 多角形
        g.fillPolygon(x, y, pt);               // 塗り潰し多角形
    }
}
```

グラフィックス

```
import java.applet.*;  
import java.awt.*;
```

```
public class ... extends Applet {  
    Graphics g;          // グラフィックスクラス
```

```
    public void paint(Graphics g) {  
        test_disp();  
    }
```

```
    public void test_disp() { //paint()以外でグラフィックスを作りまとめて描画  
        g = getGraphics();    // グラフィックス取得メソッド  
        g.drawLine(10, 10, 50, 70);  
        g.fillOval(60, 30, 15, 20);
```

```
        repaint();           // アプレット強制再描画
```

```
    }  
}
```

スレッド

```
import java.applet.*;
import java.awt.*;

public class ... extends Applet implements Runnable {
    Thread thd = null;

    public void init() {                // 初期化处理
        thd = new Thread(this);        // 自クラスでスレッド利用
        thd.start();                   // スレッド開始
    }

    public void run() {                // スレッド開始時に呼び出されるメソッド

        repaint();                     // アプレット強制再描画
    }
}
```

スレッド（一時停止）

```
import java.applet.*;
import java.awt.*;

public class ... extends Applet implements Runnable {
    Thread thd = null;

    public void init() {          // 初期化处理
        thd = new Thread(this);  // 自クラスでスレッド利用
        thd.start();             // スレッド開始
    }

    public void run() {          // スレッド開始時に呼び出されるメソッド

        repaint();              // アプレット強制再描画

        try {
            thd.sleep(500);      // スレッド一時停止(例:500ミリ秒停止)
        } catch (InterruptedException e) { }

    }
}
```

イベント処理: マウス使用

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class ... extends Applet implements MouseListener {
    public void init() {          // 初期化处理
        addMouseListener(this); // MouseEventを自クラスで受け取る宣言
    }
    // マウスがアプレット領域内へ入った時の処理 (使わなくても必要)
    public void mouseEntered(MouseEvent e) { }
    // マウスがアプレット領域外へ出た時の処理 (使わなくても必要)
    public void mouseExited(MouseEvent e) { }
    // マウス・ボタン押下時処理 (使わなくても必要)
    public void mousePressed(MouseEvent e) { }
    // マウス・ボタン押下状態→放した時の処理 (使わなくても必要)
    public void mouseReleased(MouseEvent e) { }
    // マウスクリック時の処理 (使わなくても必要)
    public void mouseClicked(MouseEvent e) {
        Point pt = e.getPoint(); // マウス位置取得メソッド
        x = pt.x; y = pt.y;
    }
}
```

イベント処理: キーボード入力

```
public class ... extends Applet implements KeyListener {  
    char key;  
    int key2;  
  
    public void init() {           // 初期化处理  
        addKeyListener(this);    // キー入力Eventを自クラス受取宣言  
    }  
  
    public void paint(Graphics g) { // 描画処理  
        requestFocusInWindow(); // アップレットフォーカスに必要  
    }  
  
    public void keyTyped(KeyEvent e) {           // キーが押された時の処理  
        key = e.getKeyChar();                   // 文字取得メソッド  
    }  
    public void keyPressed(KeyEvent e) {         // キーが押された時の処理  
        key2 = e.getKeyCode();                  // 文字code取得メソッド  
    }  
    public void keyReleased(KeyEvent e) { }      // キーが離された時の処理  
}
```

ボタン, アクションイベント

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class ... extends Applet implements ActionListener{
    Button btn;                // ボタン : 宣言

    public void init() { // 初期化处理
        btn = new Button("...");
        add(btn);
        btn.addActionListener(this); // ActionEvent自クラス受取宣言
    }

    public void actionPerformed(ActionEvent e) { // ActionEvent実行処理
        if (e.getSource() == btn) { // ボタンが押されたら...

        }
    }
}
```

アプレット領域のサイズ取得

```
import java.applet.*;
import java.awt.*;

public class ... extends Applet {
    Dimension siz;           // サイズ : 宣言
    int app_wid, app_hei;

    public void init() {     // 初期化处理
        siz = getSize();     // アプレット画面の大きさ取得
        app_wid = siz.width;  // アプレット領域の幅取得
        app_hei = siz.height; // アプレット領域の高さ取得
    }
}
```