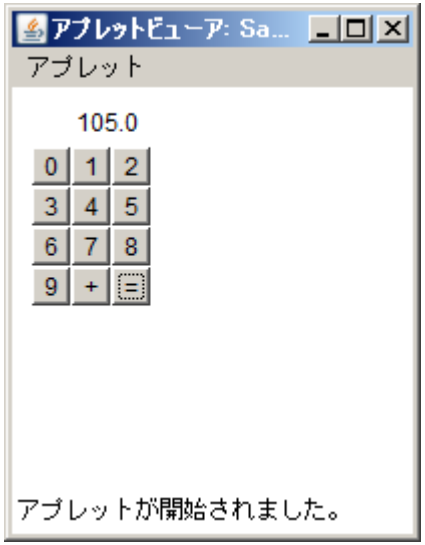
情報処理演習１０

鎌原淳三 kamahara@port.kobe-u.ac.jp

前回の課題

- ①足し算が動作プログラムとせよ
- ②ボタンの配置を電卓みたいに並べよ
- 追加課題：引き算や掛け算・割り算もできるようにする



解答

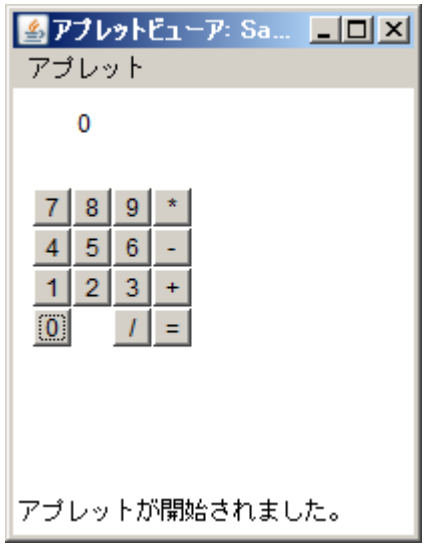
```
public void actionPerformed(ActionEvent arg0) {
    if(arg0.getActionCommand().equals("+")) {
        remNumber = dispNumber;
        operator = arg0.getActionCommand();
        dispNumber = 0;
    } else if (arg0.getActionCommand().equals("=")) {
        Double answer = remNumber + dispNumber;
            lb.setText(answer.toString());
        remNumber = answer;
        dispNumber = 0;
    } else {
        String numstr = lb.getText();
        if(numstr.equals("0")||(dispNumber==0)) {
            lb.setText(arg0.getActionCommand());
        } else {
            lb.setText(numstr+arg0.getActionCommand());
        }
        dispNumber = Double.parseDouble(lb.getText());
    }
}
```

電卓のボタン配置を変える

```
String[] num = { "0", "1", "2", "3", "4", "5",
        "6", "7", "8", "9", "+", "-", "*", "/", "=" };
int[][] pos = {
        { 10,110 }, { 10, 90 }, {30, 90 },
        { 50, 90 }, { 10, 70 }, {30, 70 },
        { 50, 70 }, { 10, 50 }, {30, 50 },
        { 50, 50 }, { 70, 90 }, {70, 70 },
        { 70, 50 }, { 50,110 }, {70,110 }
};
```

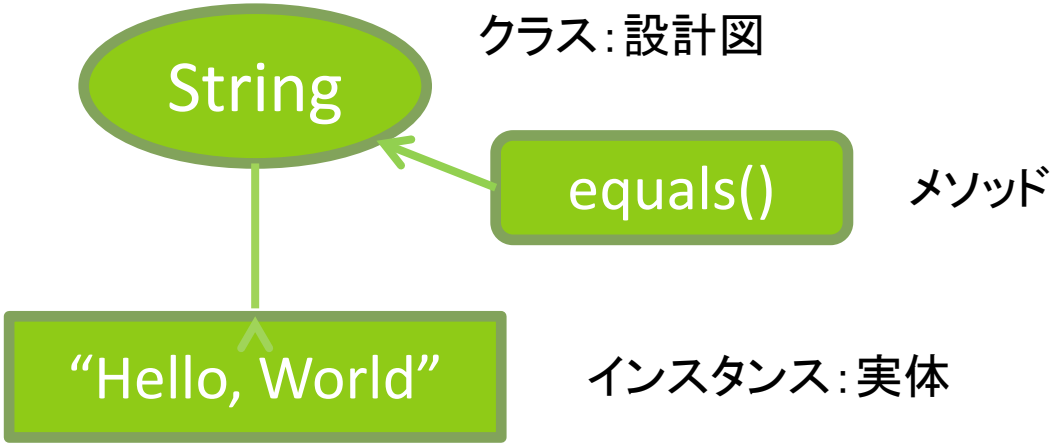
```
public double doCalc(String op, double rem, double disp) {
    if(op.equals("+")) {
        return rem+disp;
    } else if(op.equals("-")) {
        return rem-disp;
    } else if(op.equals("*")) {
        return rem*disp;
    } else {
        if(disp!=0.0) {
            return rem/disp;
        } else {
            return rem;
        }
    }
}

@Override
public void actionPerformed(ActionEvent arg0) {
    String com = arg0.getActionCommand();
    if(com.equals("+")||com.equals("-")||com.equals("*")||com.equals("/")) {
        operator = arg0.getActionCommand();
        remNumber = dispNumber;
        dispNumber = 0;
    } else if (arg0.getActionCommand().equals("=")) {
        if(operator!=null) {
            Double answer = doCalc(operator, remNumber, dispNumber);
            lb.setText(answer.toString());
            dispNumber = answer;
            //remNumber = answer;
            //dispNumber = 0;
        }
    } else {
        String numstr = lb.getText();
        if(numstr.equals("0")||(dispNumber==0)) {
```

実行例



オブジェクト指向

- オブジェクトを中心とするプログラミング方法
- Stringもクラス



クラスの定義を文書にしたもの

http://java.sun.com/javase/ja/6/docs/ja/api/java/lang/String.html

java.lang
クラス String

java.lang.Object
  └java.lang.String

すべての実装されたインタフェース:
Serializable, CharSequence, Comparable<String>

---

public final class **String**
extends Object
implements Serializable, Comparable<String>, CharSequence

String クラスは文字列を表します。Java プログラム内の「abc」などのリテラル文字列はすべて、このクラスのインスタンスとして実行されます。

文字列は定数です。この値を作成したあとに変更はできません。文字列バッファーは可変文字列をサポートします。文字列オブジェクトは不変であるため、共用することができます。例を示します。

```
String str = "abc";
```

は、次と同じです。

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

文字列がどのように使われるかについて、さらに例を示します。

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

メソッドの一覧

メソッドの概要	
char	charAt(int index) 指定されたインデックス位置にある char 値を返します。
int	codePointAt(int index) 指定されたインデックス位置の文字 (Unicode コードポイント) を返します。
int	codePointBefore(int index) 指定されたインデックスの前の文字 (Unicode コードポイント) を返します。
int	codePointCount(int beginIndex, int endIndex) この String の指定されたテキスト範囲の Unicode コードポイントの数を返します。
int	compareTo(String anotherString) 2 つの文字列を辞書的に比較します。
int	compareToIgnoreCase(String str) 大文字と小文字の区別なしで、2 つの文字列を辞書的に比較します。
String	concat(String str) 指定された文字列をこの文字列の最後に連結します。
boolean	contains(CharSequence s) この文字列が指定された char 値のシーケンスを含む場合に限り true を返します。
boolean	contentEquals(CharSequence cs) この文字列と指定された CharSequence を比較します。
boolean	contentEquals(StringBuffer sb) この文字列と指定された StringBuffer を比較します。
static String	copyValueOf(char[] data) 指定された配列内の文字シーケンスを表す String を返します。
static String	copyValueOf(char[] data, int offset, int count) 指定された配列内の文字シーケンスを表す String を返します。
boolean	endsWith(String suffix) この文字列が、指定された接尾辞で終るかどうかを判定します。
boolean	equals(Object anObject) この文字列と指定されたオブジェクトを比較します。

EQUALS()の説明

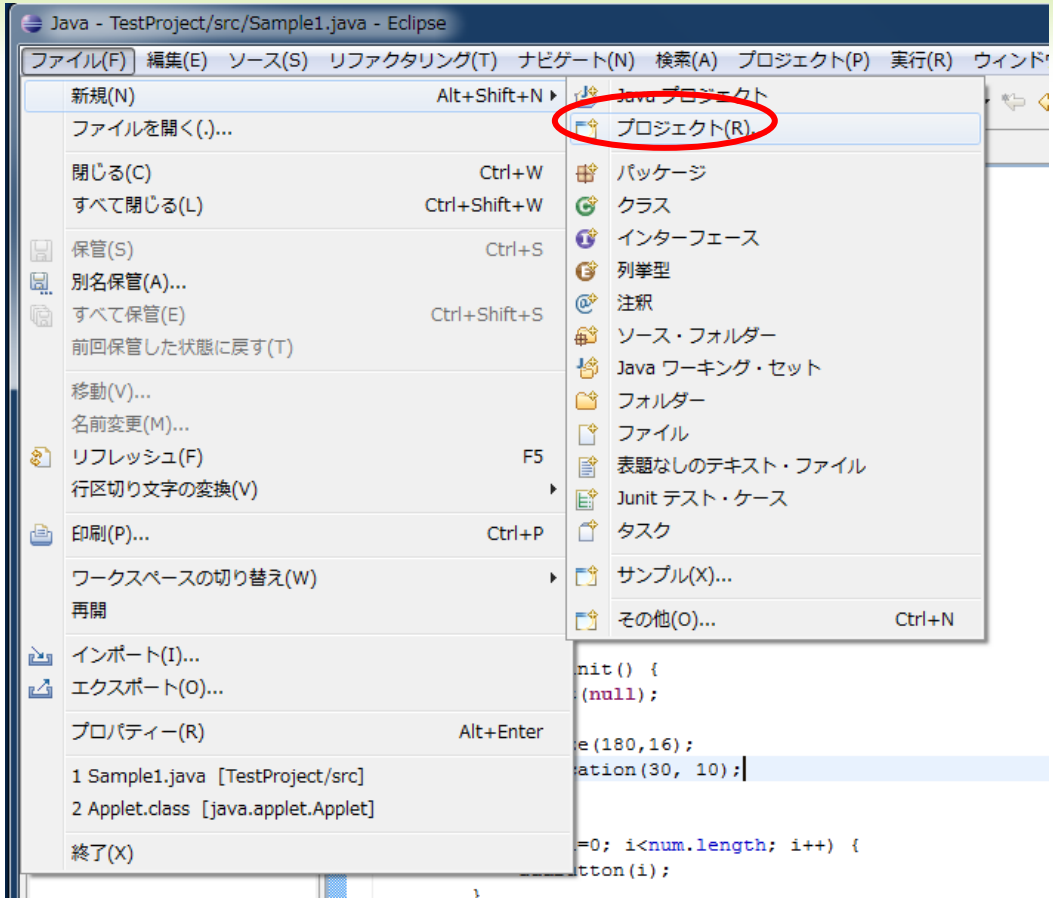
equals

public boolean **equals**(Object anObject)

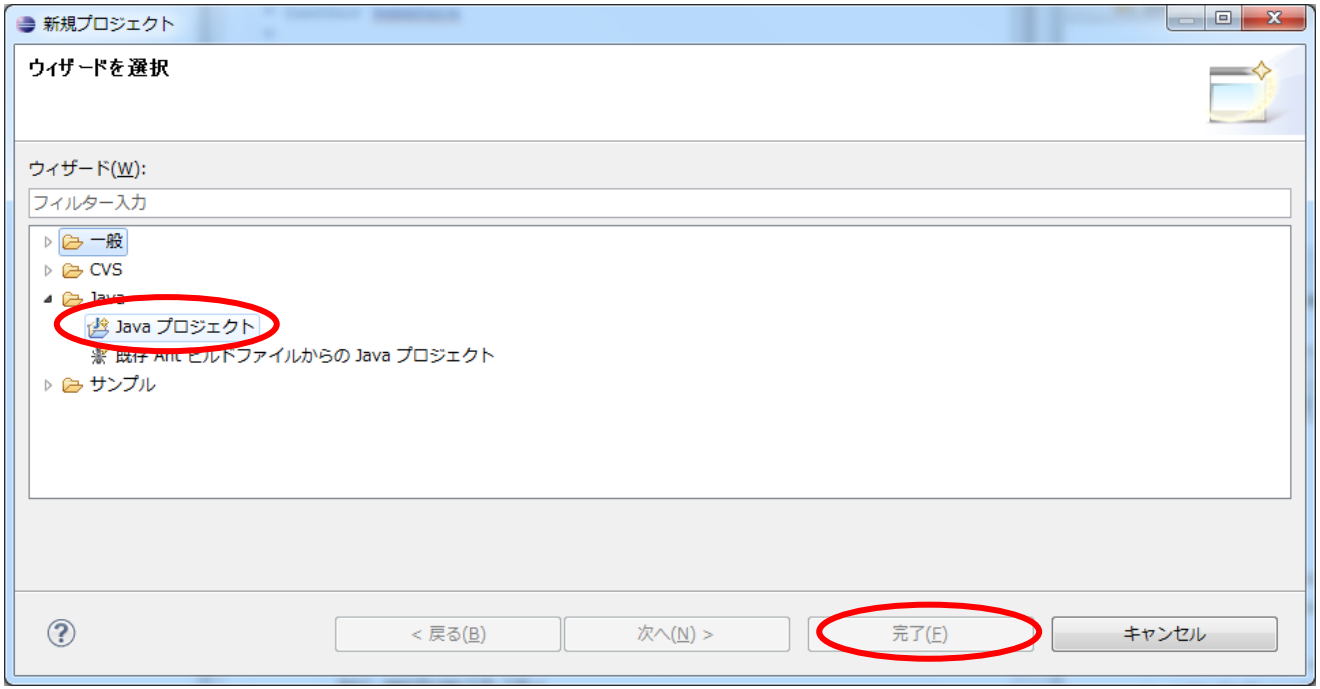
この文字列と指定されたオブジェクトを比較します。引数が null でなく、このオブジェクトと同じ文字シーケンスを表す String オブジェクトである場合にだけ、結果は true になります。

オーバーライド:
クラス Object 内の equals
パラメータ:
anObject - この String と比較されるオブジェクト
戻り値:
指定されたオブジェクトがこの文字列に等しい String を表す場合は true、そうでない場合は false
関連項目:
compareTo(String), equalsIgnoreCase(String)

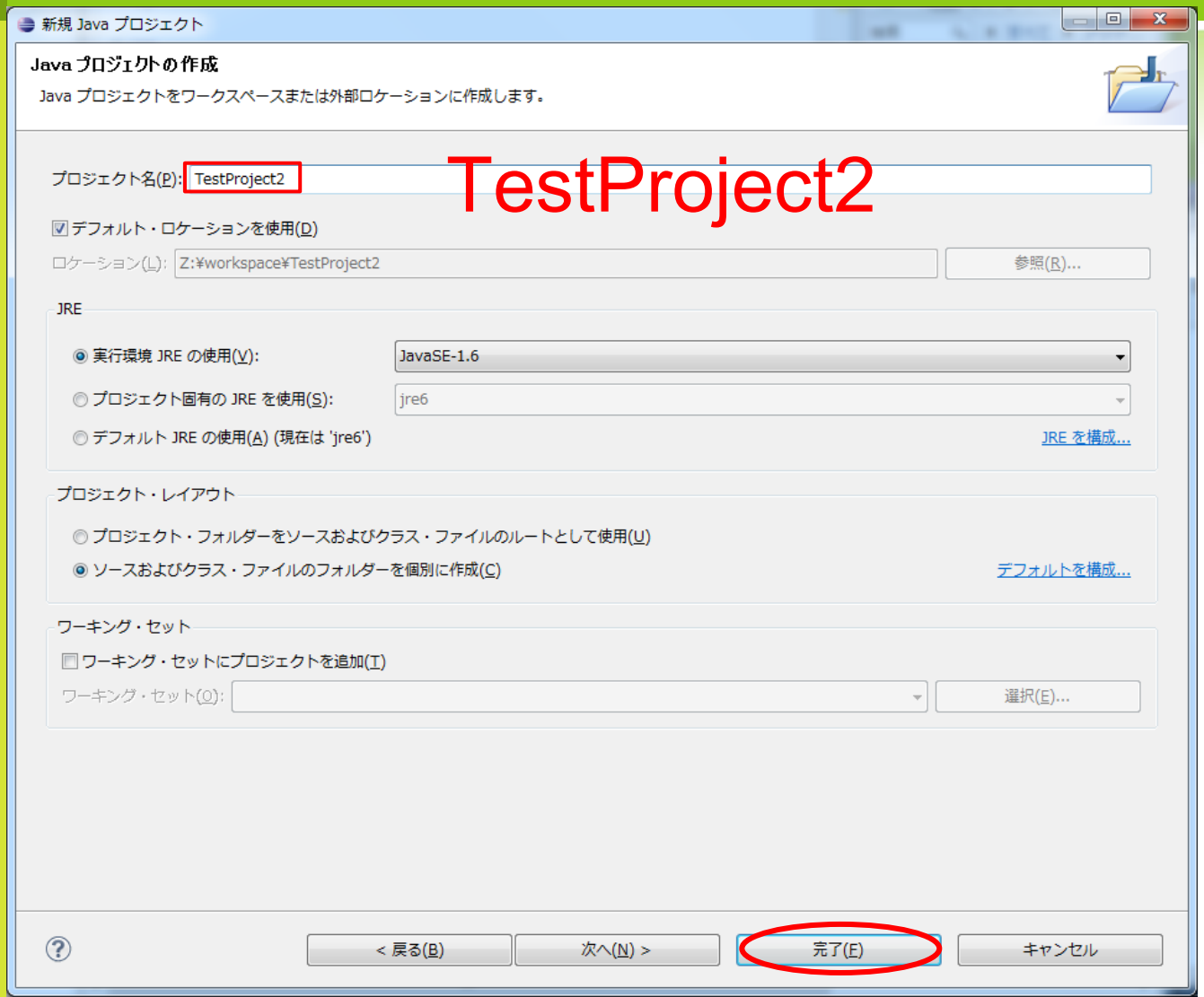
新しいプロジェクトを作る



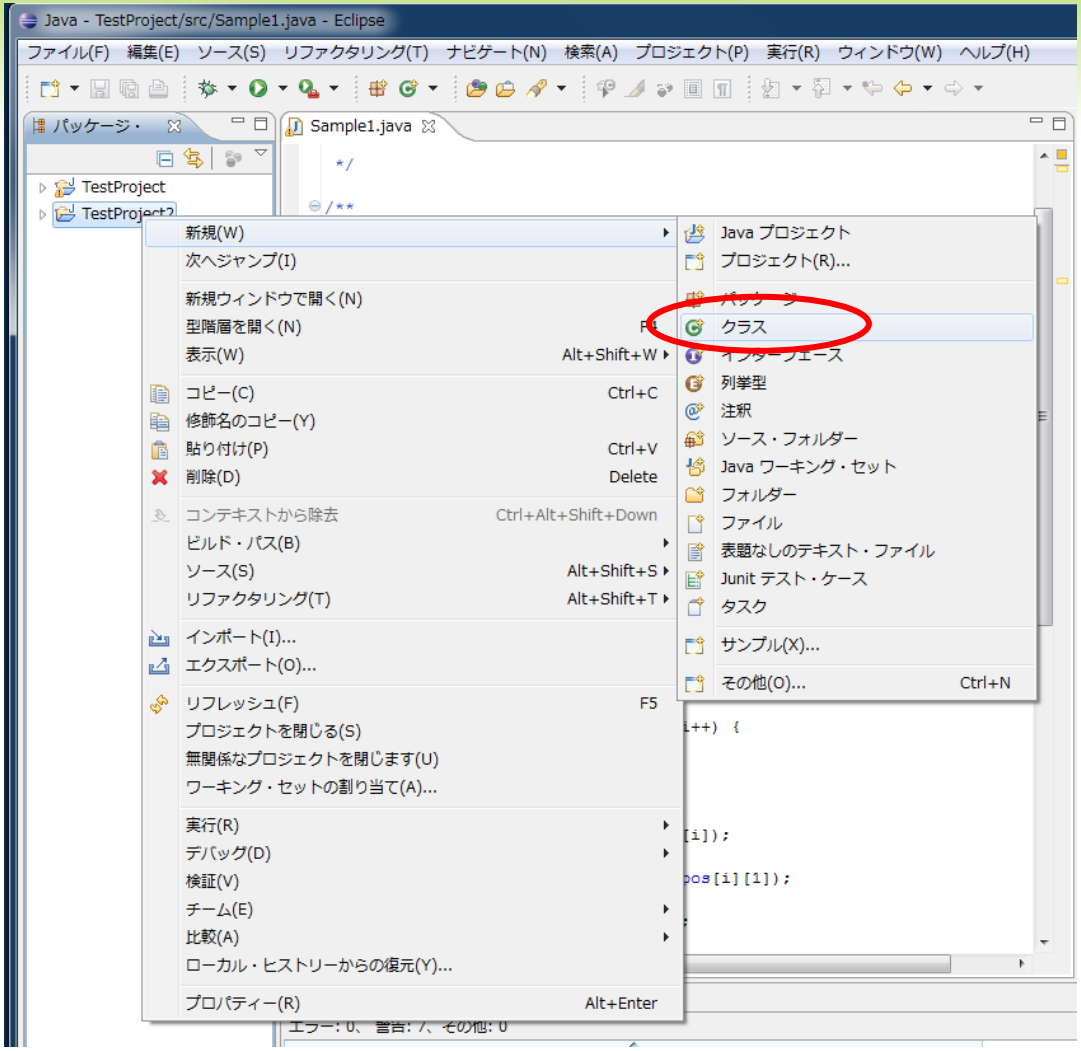
Javaプロジェクトを指定



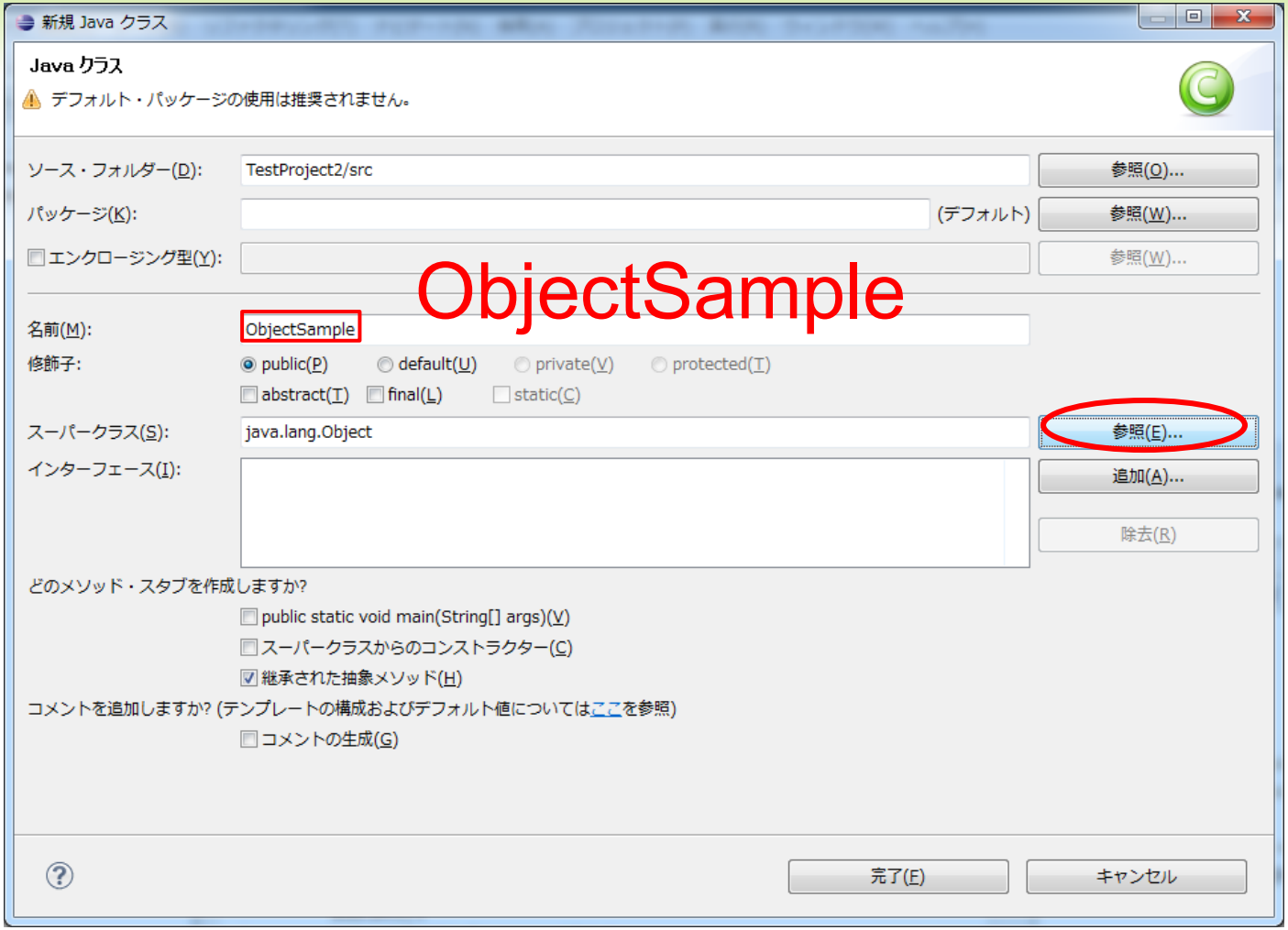
プロジェクト名を指定


新規 Java プロジェクト
Java プロジェクトの作成
Java プロジェクトをワークスペースまたは外部ロケーションに作成します。
プロジェクト名(P): TestProject2
TestProject2
デフォルト・ロケーションを使用(D)
ロケーション(L): Z:¥workspace¥TestProject2
参照(R)...
JRE
実行環境 JRE の使用(V):
JavaSE-1.6
プロジェクト固有の JRE を使用(S):
jre6
デフォルト JRE の使用(A) (現在は 'jre6')
JRE を構成...
プロジェクト・レイアウト
プロジェクト・フォルダーをソースおよびクラス・ファイルのルートとして使用(U)
ソースおよびクラス・ファイルのフォルダーを個別に作成(C)
デフォルトを構成...
ワーキング・セット
ワーキング・セットにプロジェクトを追加(T)
ワーキング・セット(O):
選択(E)...
< 戻る(B)
次へ(N) >
完了(F)
キャンセル

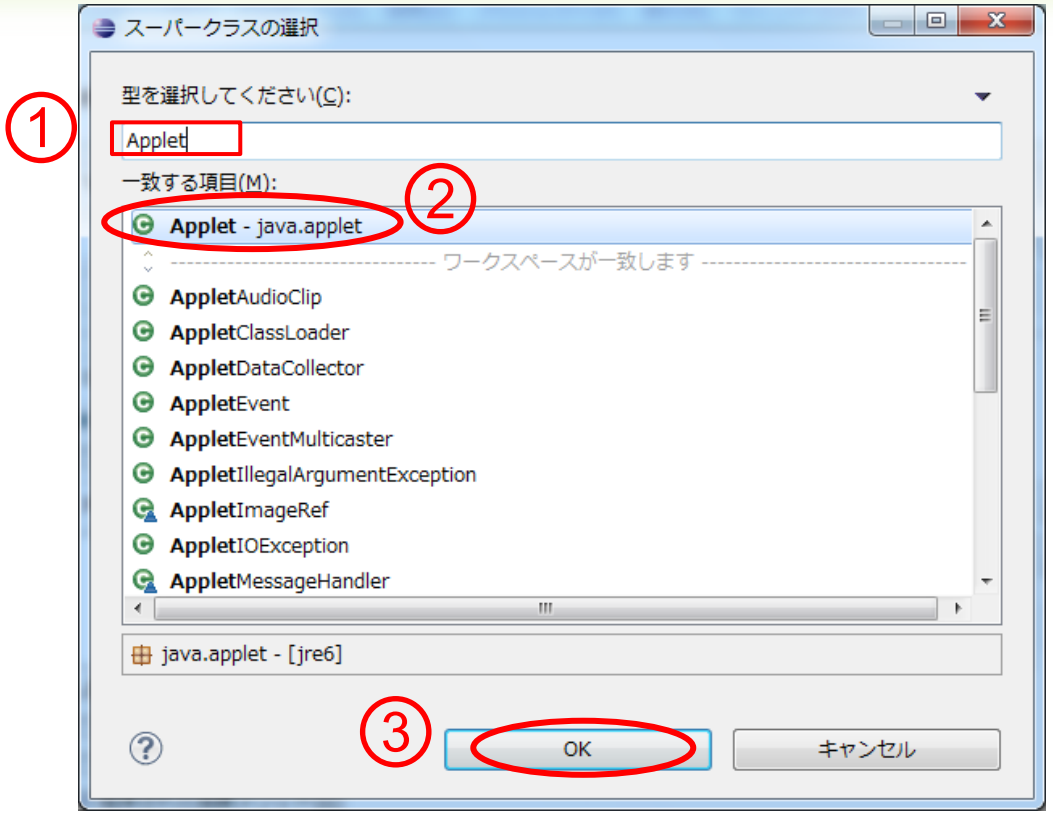
プロジェクト名の上でクラスを新規作成



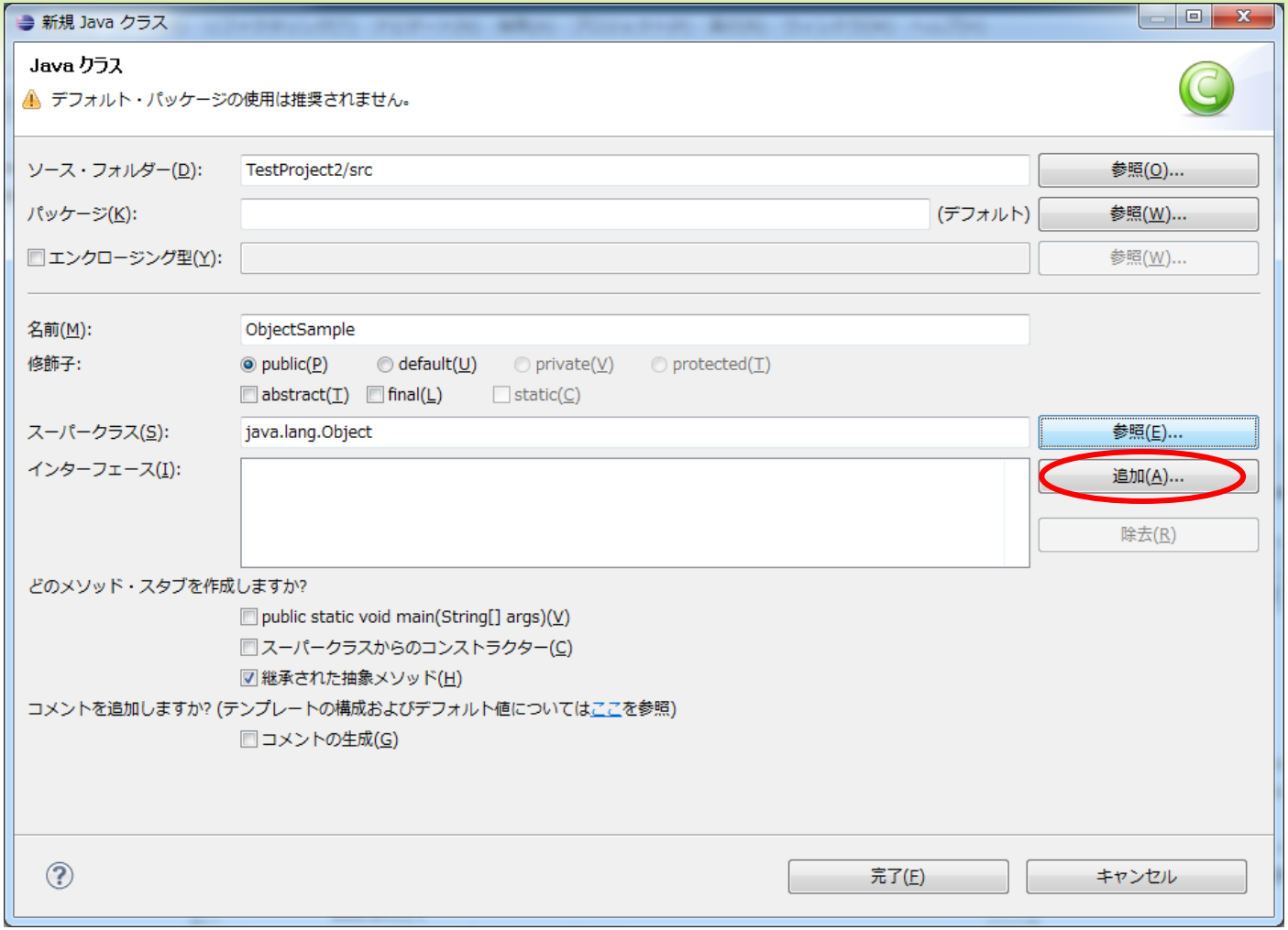
クラス名を入力


新規 Java クラス
Java クラス
デフォルト・パッケージの使用は推奨されません。
ソース・フォルダー(D):
TestProject2/src
参照(O)...
パッケージ(K):
(デフォルト)
参照(W)...
エンクロージング型(Y):
参照(W)...
ObjectSample
名前(M):
ObjectSample
修飾子:
public(P)
default(U)
private(V)
protected(T)
abstract(T)
final(L)
static(C)
スーパークラス(S):
java.lang.Object
参照(E)...
インターフェース(I):
追加(A)...
除去(R)
どのメソッド・スタブを作成しますか?
public static void main(String[] args)(V)
スーパークラスからのコンストラクター(C)
継承された抽象メソッド(H)
コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照)
コメントの生成(G)
完了(F)
キャンセル

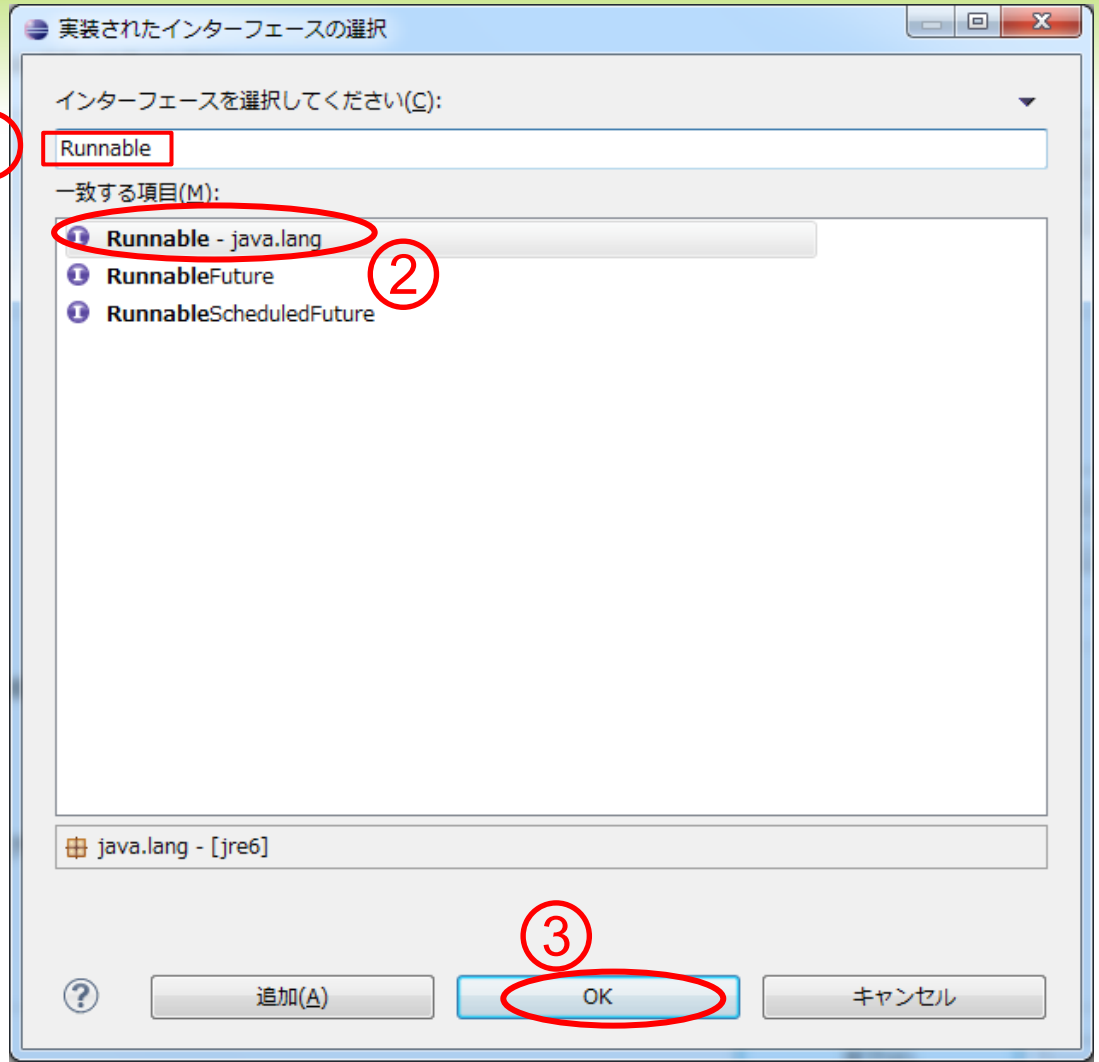
アプレットを入力して指定



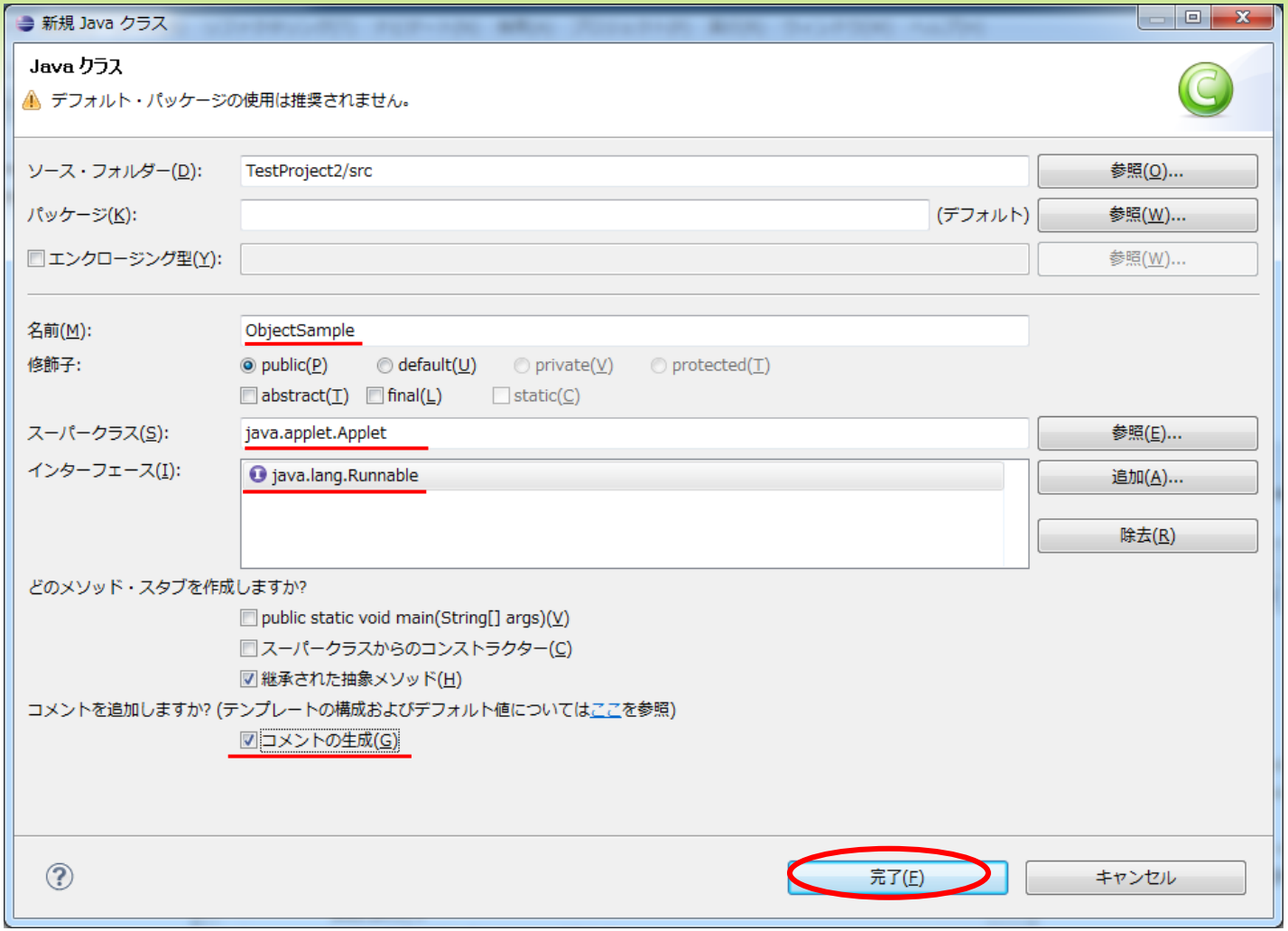
インターフェースを追加



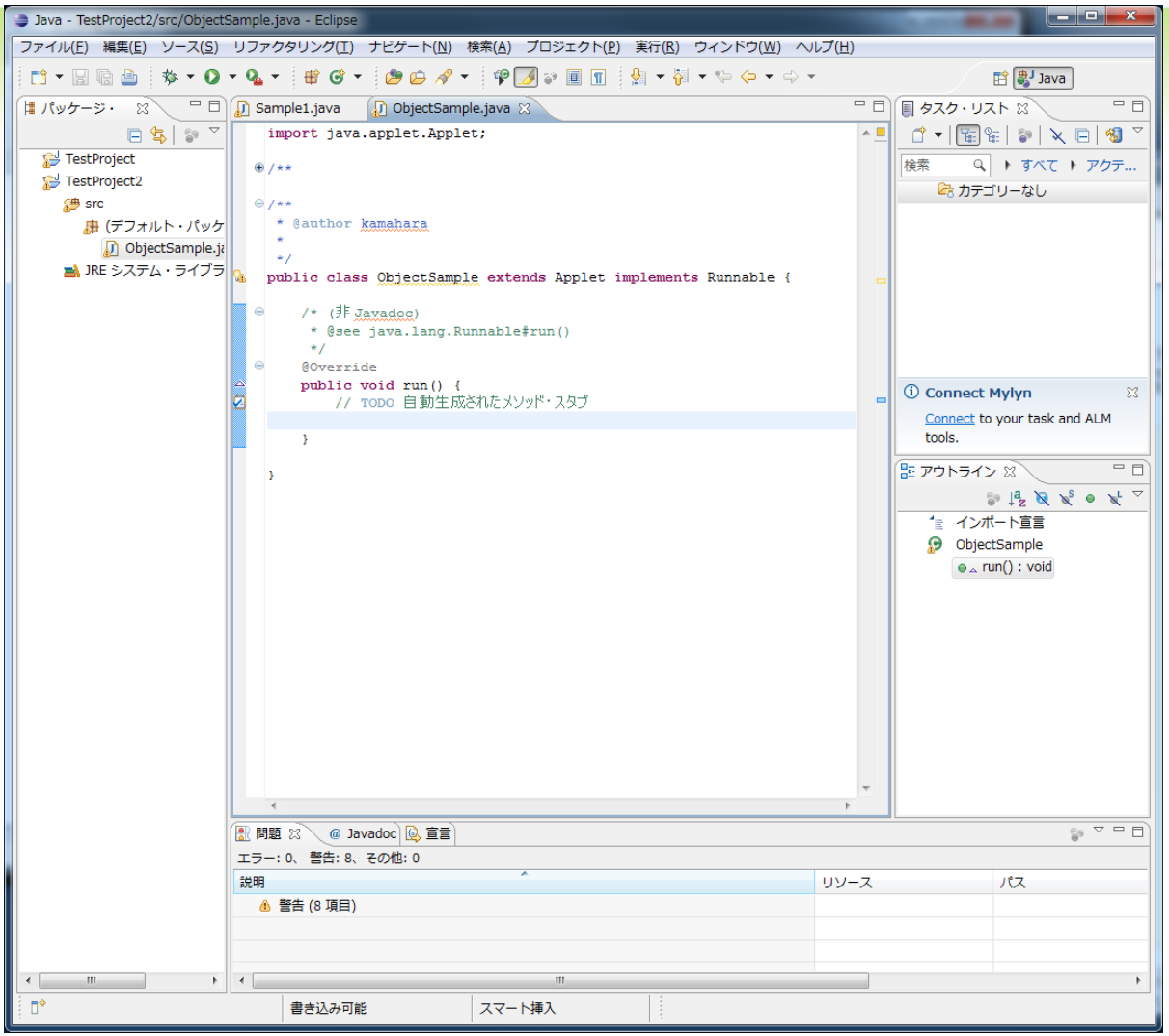
RUNNABLEを指定する



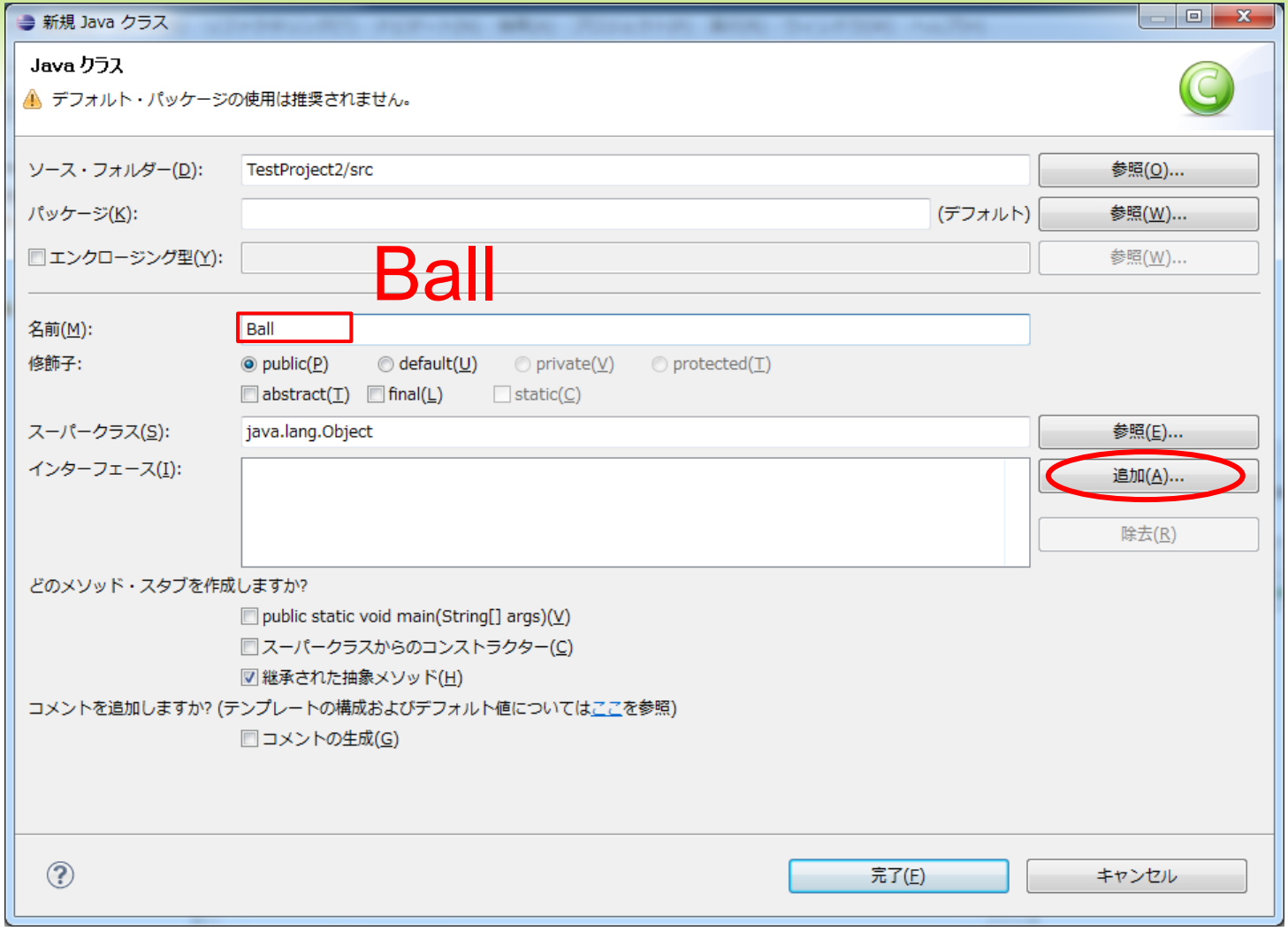
下線部を確認


新規 Java クラス
Java クラス
デフォルト・パッケージの使用は推奨されません。
ソース・フォルダー(D):
TestProject2/src
参照(O)...
パッケージ(K):
(デフォルト)
参照(W)...
エンクロージング型(Y):
参照(W)...
名前(M):
ObjectSample
修飾子:
public(P)
default(U)
private(V)
protected(T)
abstract(T)
final(L)
static(C)
スーパークラス(S):
java.applet.Applet
参照(E)...
インターフェース(I):
java.lang.Runnable
追加(A)...
除去(R)
どのメソッド・スタブを作成しますか?
public static void main(String[] args)(V)
スーパークラスからのコンストラクター(C)
継承された抽象メソッド(H)
コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照)
コメントの生成(G)
完了(F)
キャンセル

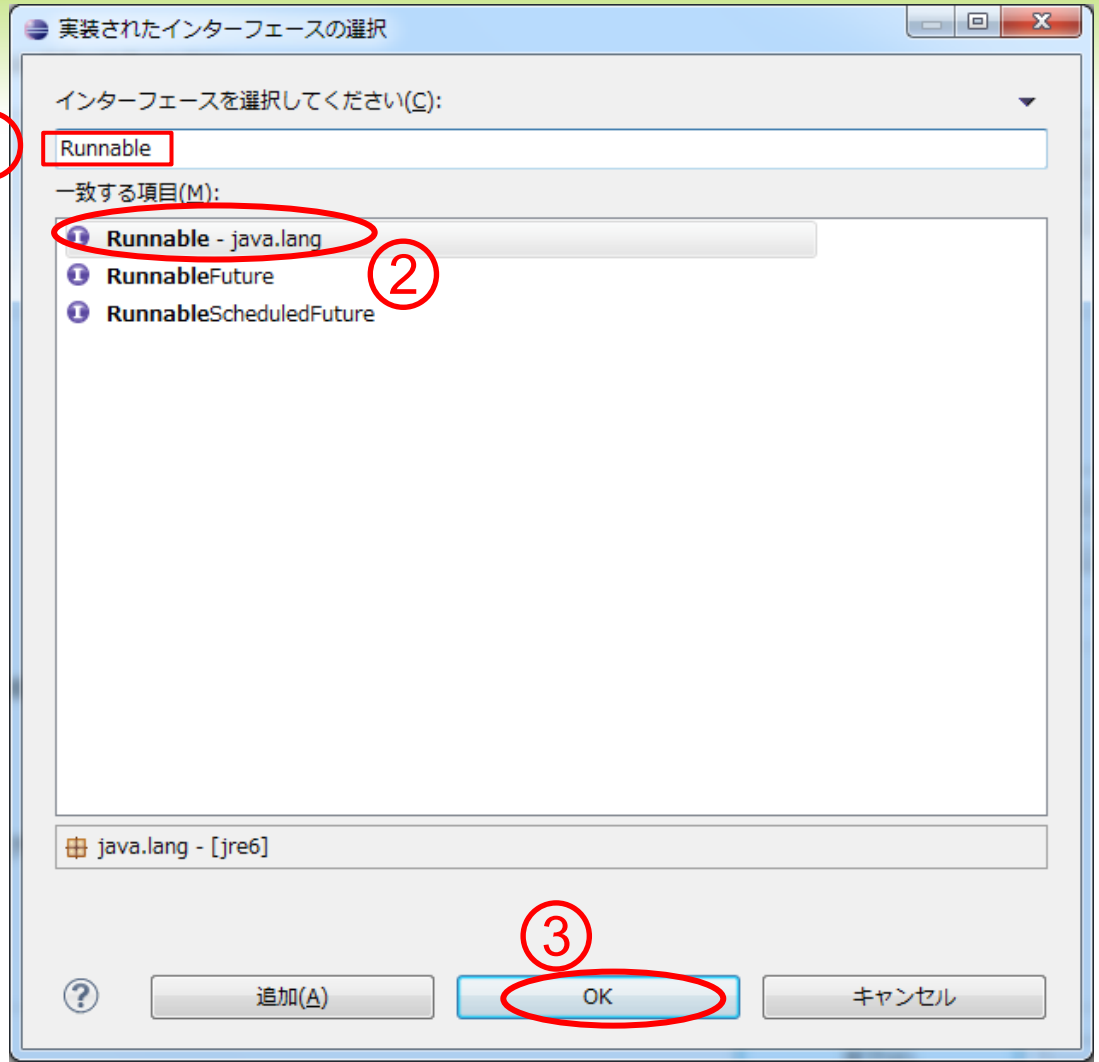
スケルトンができた


Java - TestProject2/src/ObjectSample.java - Eclipse
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
パッケージ・
TestProject
TestProject2
src
(デフォルト・パッケ
ObjectSample.ja
JRE システム・ライブラ
Sample1.java
ObjectSample.java
import java.applet.Applet;
/**
/**
* @author kamahara
*
*/
public class ObjectSample extends Applet implements Runnable {
/* (非 Javadoc)
* @see java.lang.Runnable#run()
*/
@Override
public void run() {
// TODO 自動生成されたメソッド・スタブ
}
}
タスク・リスト
検索
すべて
アクテ...
カテゴリーなし
Connect Mylyn
Connect to your task and ALM tools.
アウトライン
インポート宣言
ObjectSample
run() : void
問題
Javadoc
宣言
エラー: 0、 警告: 8、その他: 0
説明
リソース
パス
警告 (8 項目)
書き込み可能
スマート挿入

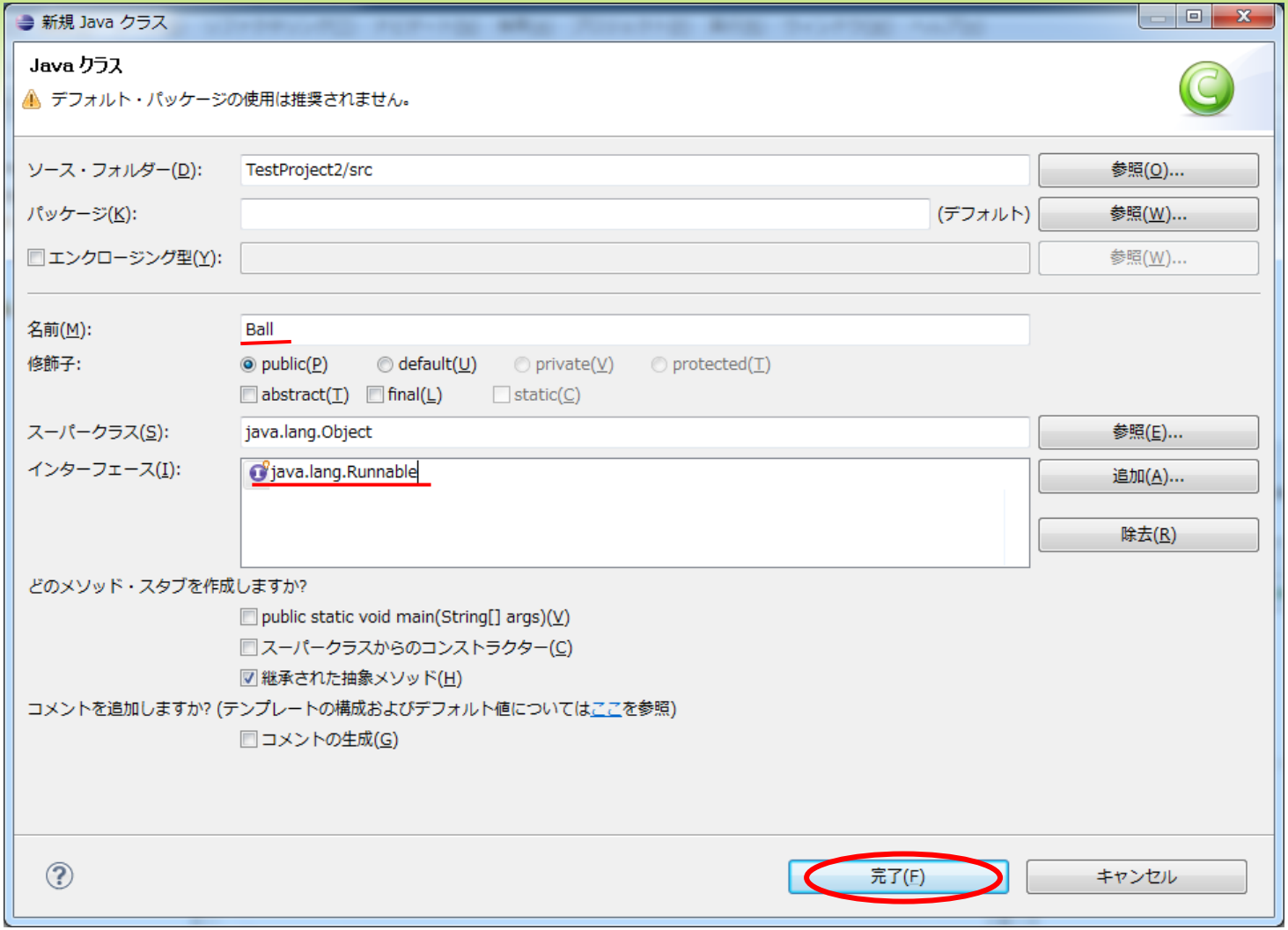
もう１つクラスを作る


新規 Java クラス
Java クラス
デフォルト・パッケージの使用は推奨されません。
ソース・フォルダー(D):
TestProject2/src
参照(O)...
パッケージ(K):
(デフォルト)
参照(W)...
エンクロージング型(Y):
参照(W)...
Ball
名前(M):
Ball
修飾子:
public(P)
default(U)
private(V)
protected(T)
abstract(T)
final(L)
static(C)
スーパークラス(S):
java.lang.Object
参照(E)...
インターフェース(I):
追加(A)...
除去(R)
どのメソッド・スタブを作成しますか?
public static void main(String[] args)(V)
スーパークラスからのコンストラクター(C)
継承された抽象メソッド(H)
コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照)
コメントの生成(G)
完了(F)
キャンセル

RUNNABLEを指定する



下線部を確認


新規 Java クラス
Java クラス
デフォルト・パッケージの使用は推奨されません。
ソース・フォルダー(D):
TestProject2/src
参照(O)...
パッケージ(K):
(デフォルト)
参照(W)...
エンクロージング型(Y):
参照(W)...
名前(M):
Ball
修飾子:
public(P)
default(U)
private(V)
protected(T)
abstract(T)
final(L)
static(C)
スーパークラス(S):
java.lang.Object
参照(E)...
インターフェース(I):
java.lang.Runnable
追加(A)...
除去(R)
どのメソッド・スタブを作成しますか?
public static void main(String[] args)(V)
スーパークラスからのコンストラクター(C)
継承された抽象メソッド(H)
コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照)
コメントの生成(G)
完了(F)
キャンセル

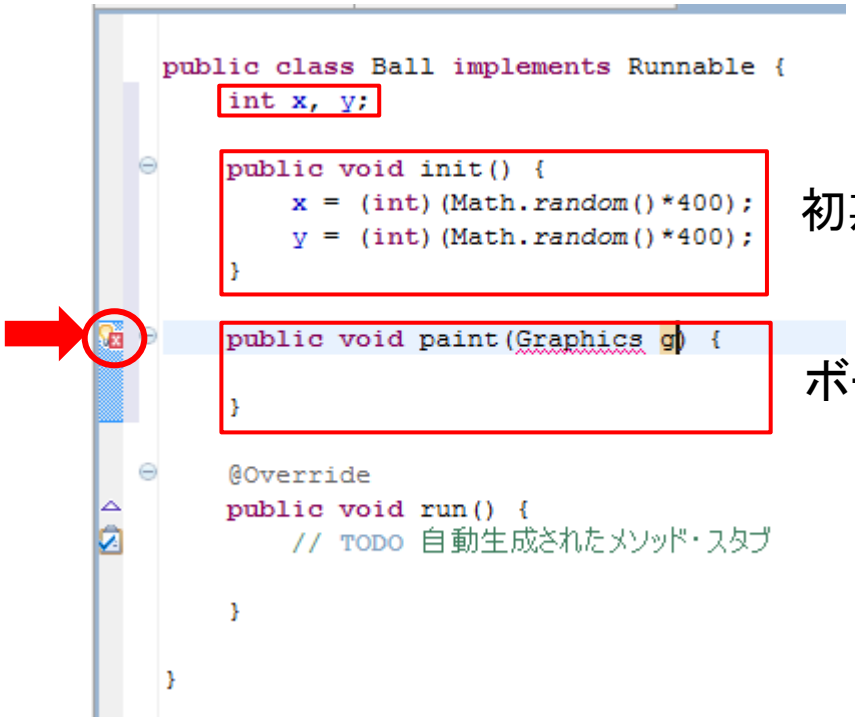
BALL.JAVAが生成された

```
public class Ball implements Runnable {

    @Override
    public void run() {
        // TODO 自動生成されたメソッド・スタブ

    }

}
```

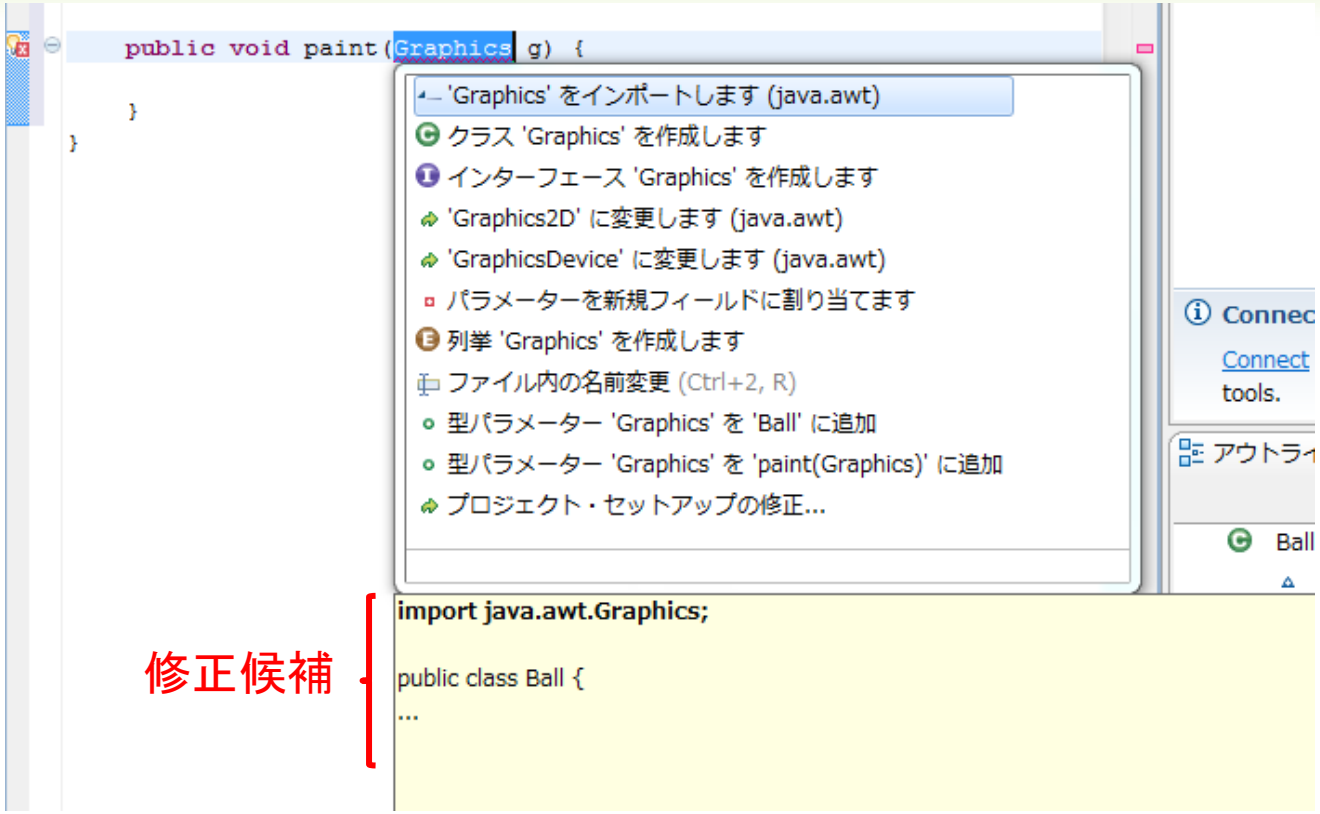
インタフェースにRunnableをつけると、run()メソッドが必要になるので、自動的に生成される

初期化メソッドを作る

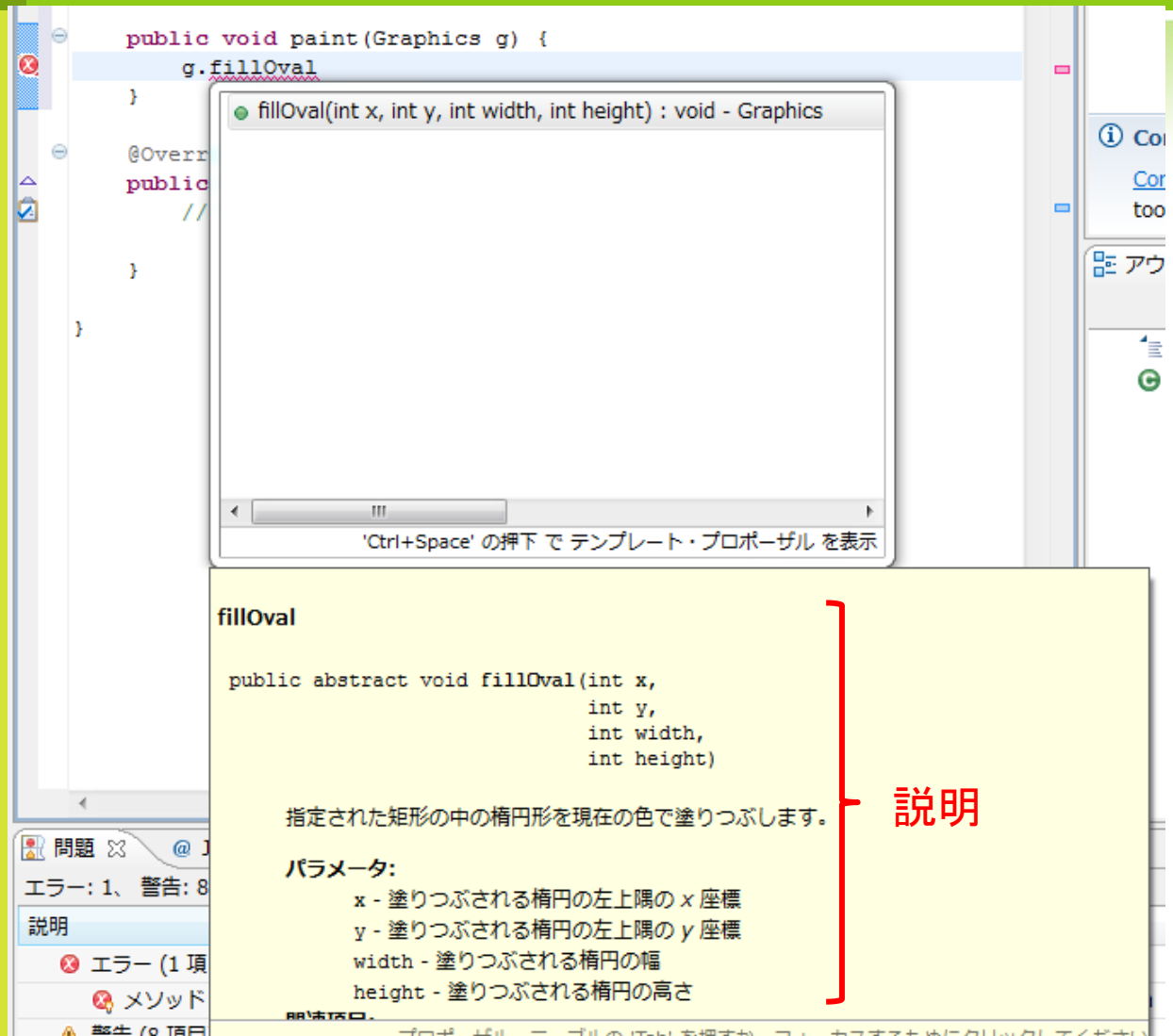
- init()でボールの初期位置のxとyをランダムに決める



GRAPHICSをインポート



FILLOVALで円を描く



5x5の円をx,yに描画する

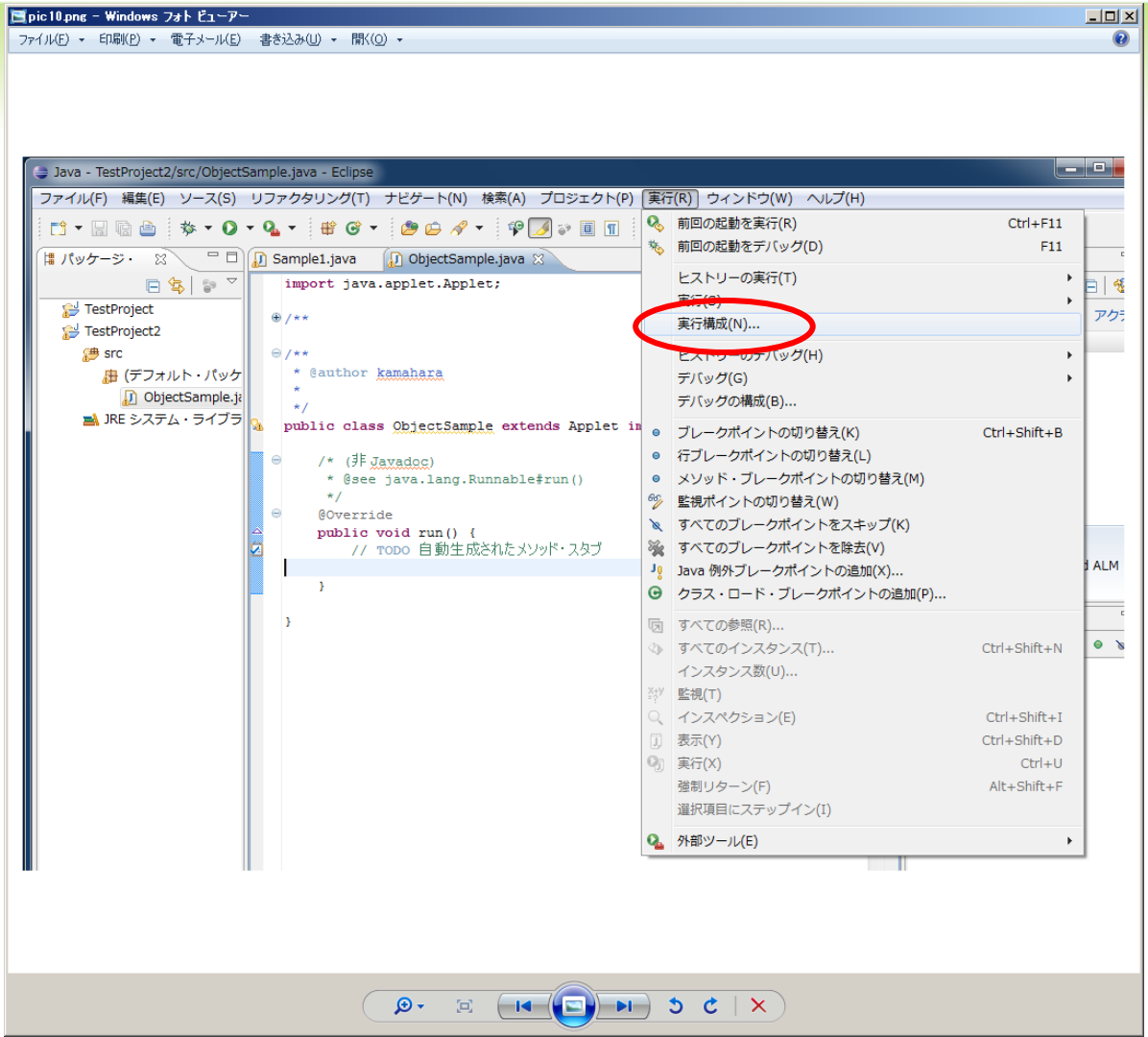
```
import java.awt.Graphics;


public class Ball implements Runnable {
    int x, y;

    public void init() {
        x = (int)(Math.random()*400);
        y = (int)(Math.random()*400);
    }

    public void paint(Graphics g) {
        g.fillOval( x, y, 5, 5);
    }

    @Override
    public void run() {
        // TODO 自動生成されたメソッド・スタブ

    }

}
```

OBJECTSAMPLE.JAVAに戻る

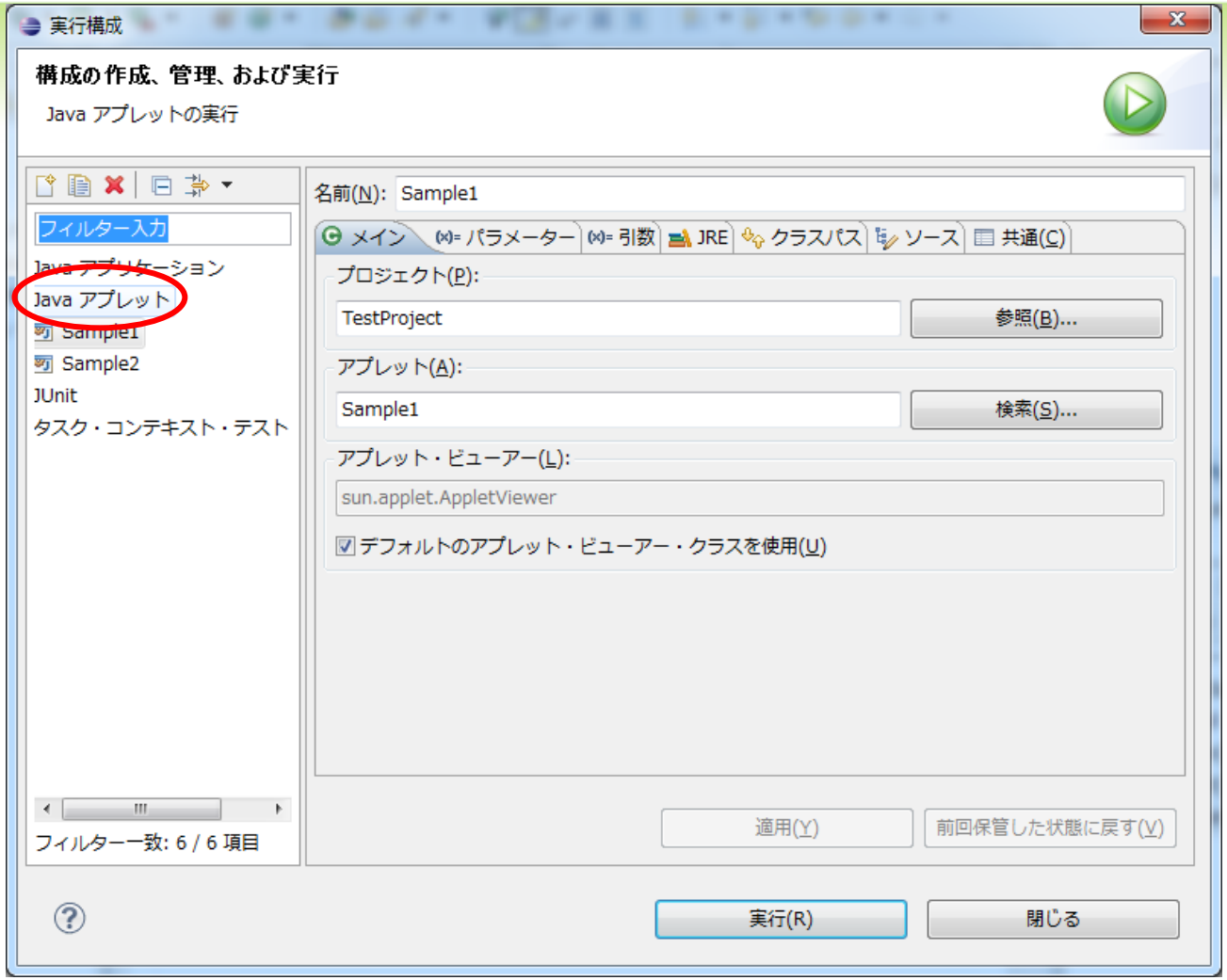
Sample1.java | ObjectSample.java | Ball.java

```
import java.applet.Applet;
import java.awt.Graphics;

/**

/**
 * @author kamahara
 *
 */
public class ObjectSample extends Applet implements Runnable {
    Ball ball;

    /* (非 Javadoc)
     * @see java.lang.Runnable#run()
     */
    @Override
    public void run() {
        // TODO 自動生成されたメソッド・スタブ

    }

    public void init() {
        ball = new Ball();
        ball.init();
    }

    public void paint(Graphics g) {
        ball.paint(g);
    }
}
```

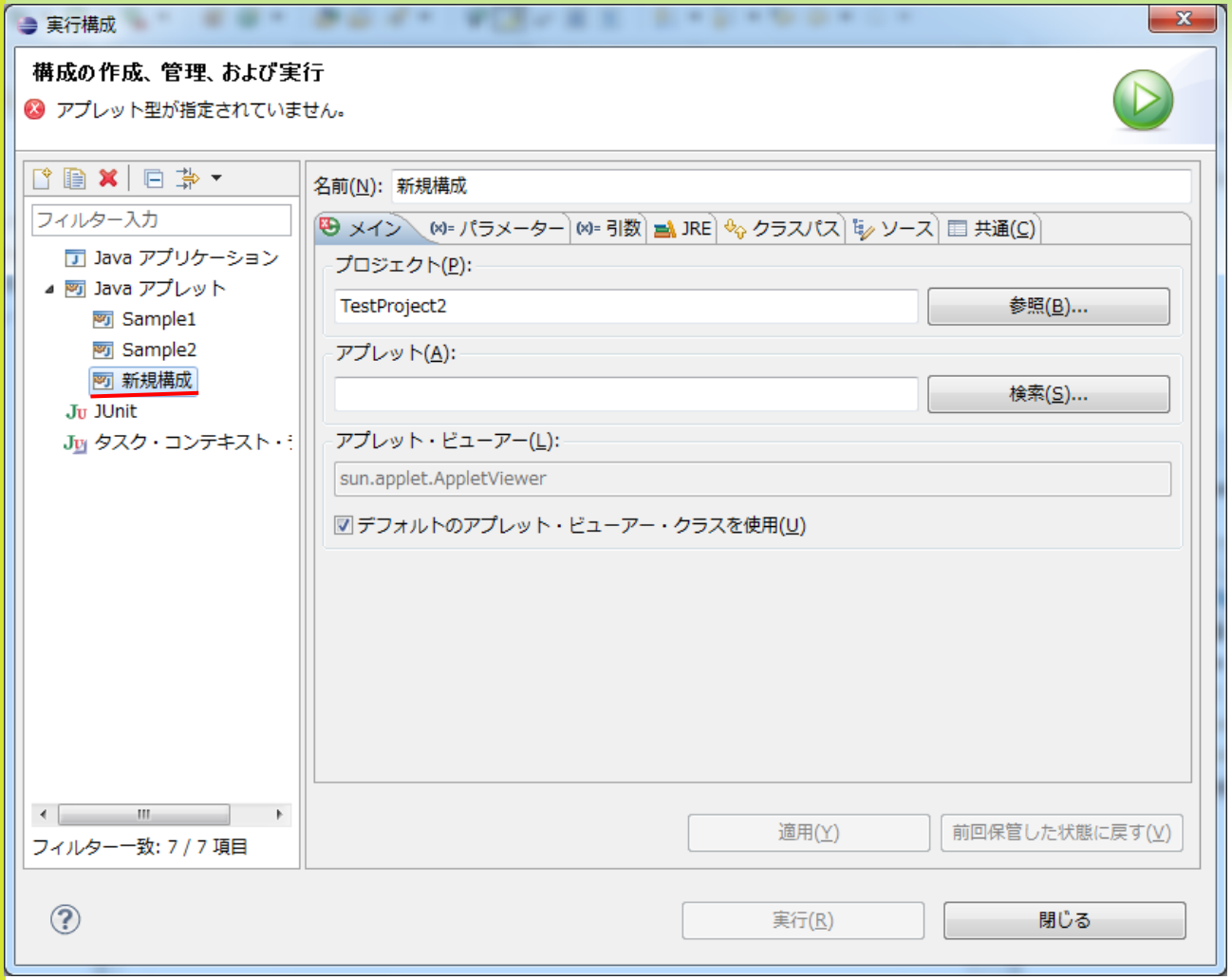
実行環境を設定する


pic10.png - Windows フォト ビューアー
ファイル(F) 印刷(P) 電子メール(E) 書き込み(U) 開く(O)
Java - TestProject2/src/ObjectSample.java - Eclipse
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
パッケージ・
TestProject
TestProject2
src
(デフォルト・パッケ
ObjectSample.ja
JRE システム・ライブラ
Sample1.java
ObjectSample.java
import java.applet.Applet;
/**
/**
* @author kamahara
*
*/
public class ObjectSample extends Applet im
/* (非 Javadoc)
* @see java.lang.Runnable#run()
*/
@Override
public void run() {
// TODO 自動生成されたメソッド・スタブ
}
}
前回の起動を実行(R) Ctrl+F11
前回の起動をデバッグ(D) F11
ヒストリーの実行(T)
実行(S)
実行構成(N)...
ヒストリーのデバッグ(H)
デバッグ(G)
デバッグの構成(B)...
ブレークポイントの切り替え(K) Ctrl+Shift+B
行ブレークポイントの切り替え(L)
メソッド・ブレークポイントの切り替え(M)
監視ポイントの切り替え(W)
すべてのブレークポイントをスキップ(K)
すべてのブレークポイントを除去(V)
Java 例外ブレークポイントの追加(X)...
クラス・ロード・ブレークポイントの追加(P)...
すべての参照(R)...
すべてのインスタンス(T)... Ctrl+Shift+N
インスタンス数(U)...
監視(T)
インスペクション(E) Ctrl+Shift+I
表示(Y) Ctrl+Shift+D
実行(X) Ctrl+U
強制リターン(F) Alt+Shift+F
選択項目にステップイン(I)
外部ツール(E)

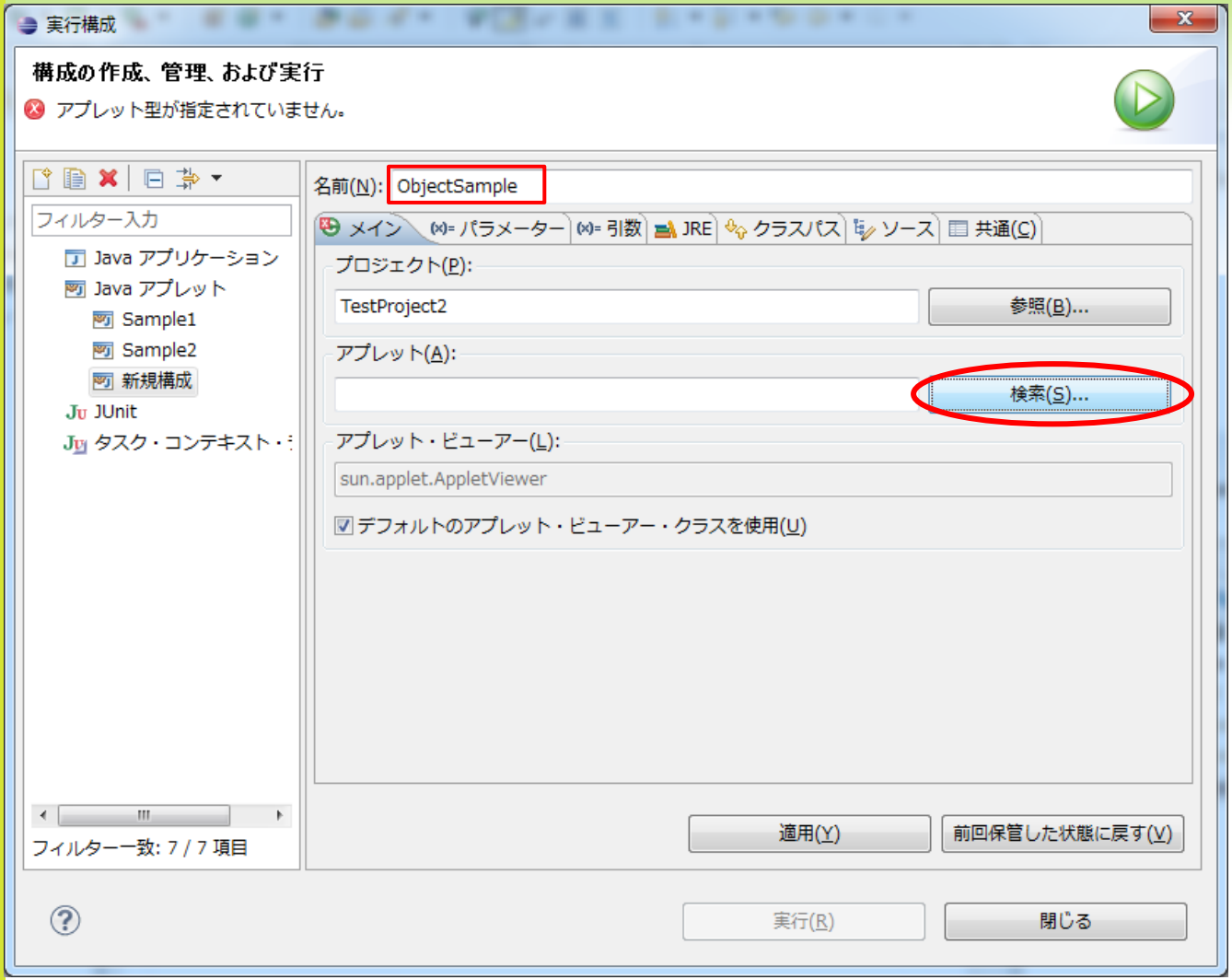
JAVAアプレットをダブルクリック



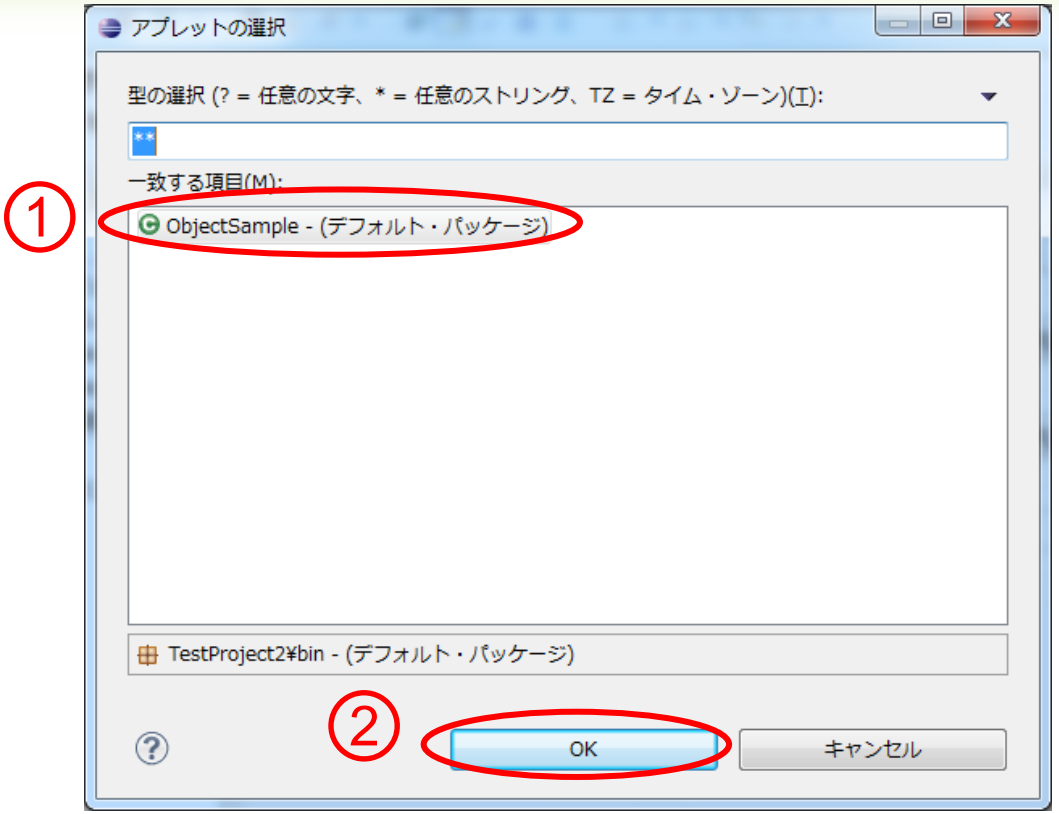
新しい「実行の構成」ができた



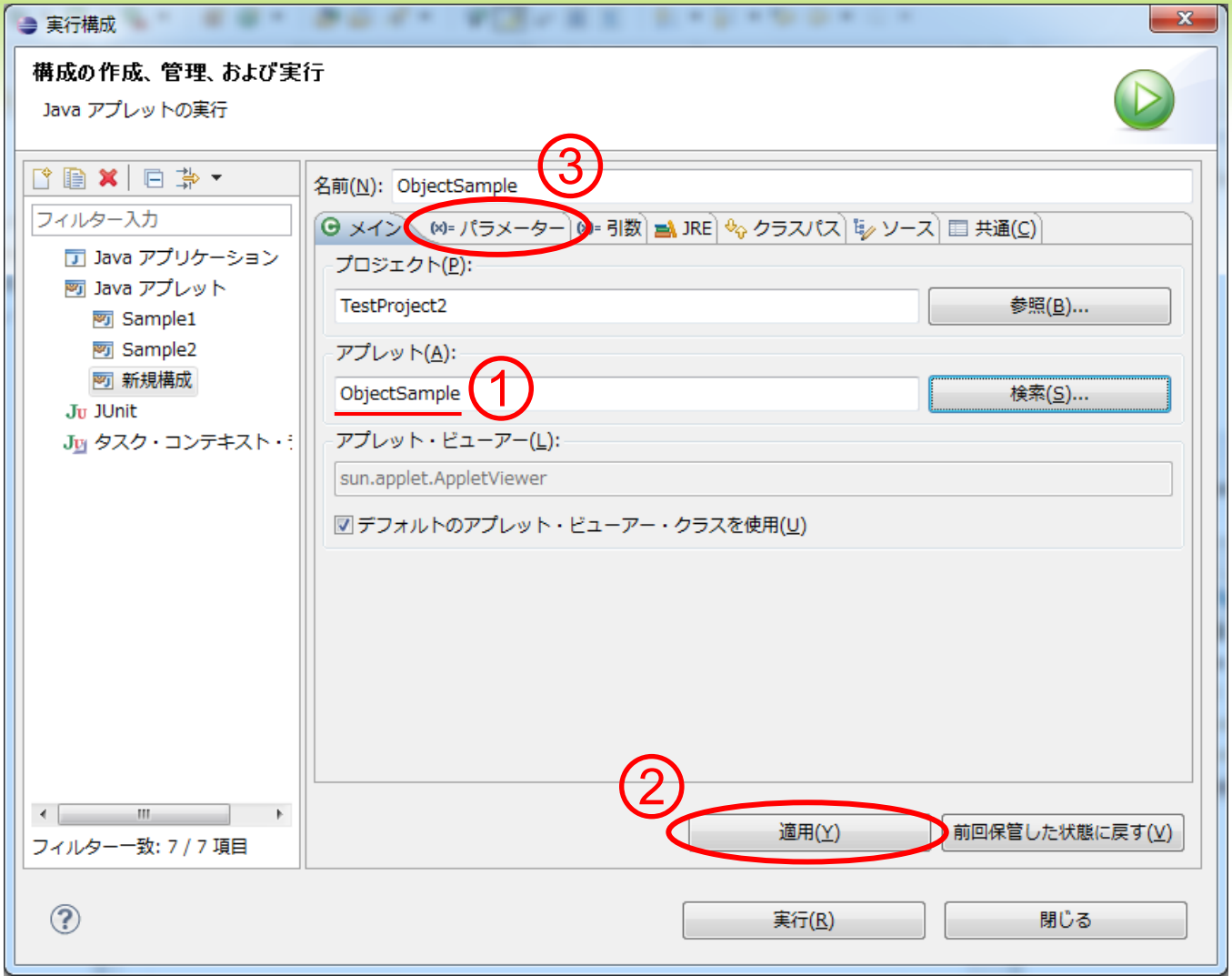
構成の名前を指定


実行構成
構成の作成、管理、および実行
アプレット型が指定されていません。
名前(N): ObjectSample
フィルター入力
Java アプリケーション
Java アプレット
Sample1
Sample2
新規構成
JUnit
タスク・コンテキスト・
フィルター一致: 7 / 7 項目
メイン
パラメーター
引数
JRE
クラスパス
ソース
共通(C)
プロジェクト(P):
TestProject2
参照(B)...
アプレット(A):
検索(S)...
アプレット・ビューアー(L):
sun.applet.AppletViewer
デフォルトのアプレット・ビューアー・クラスを使用(U)
適用(Y)
前回保管した状態に戻す(V)
実行(R)
閉じる

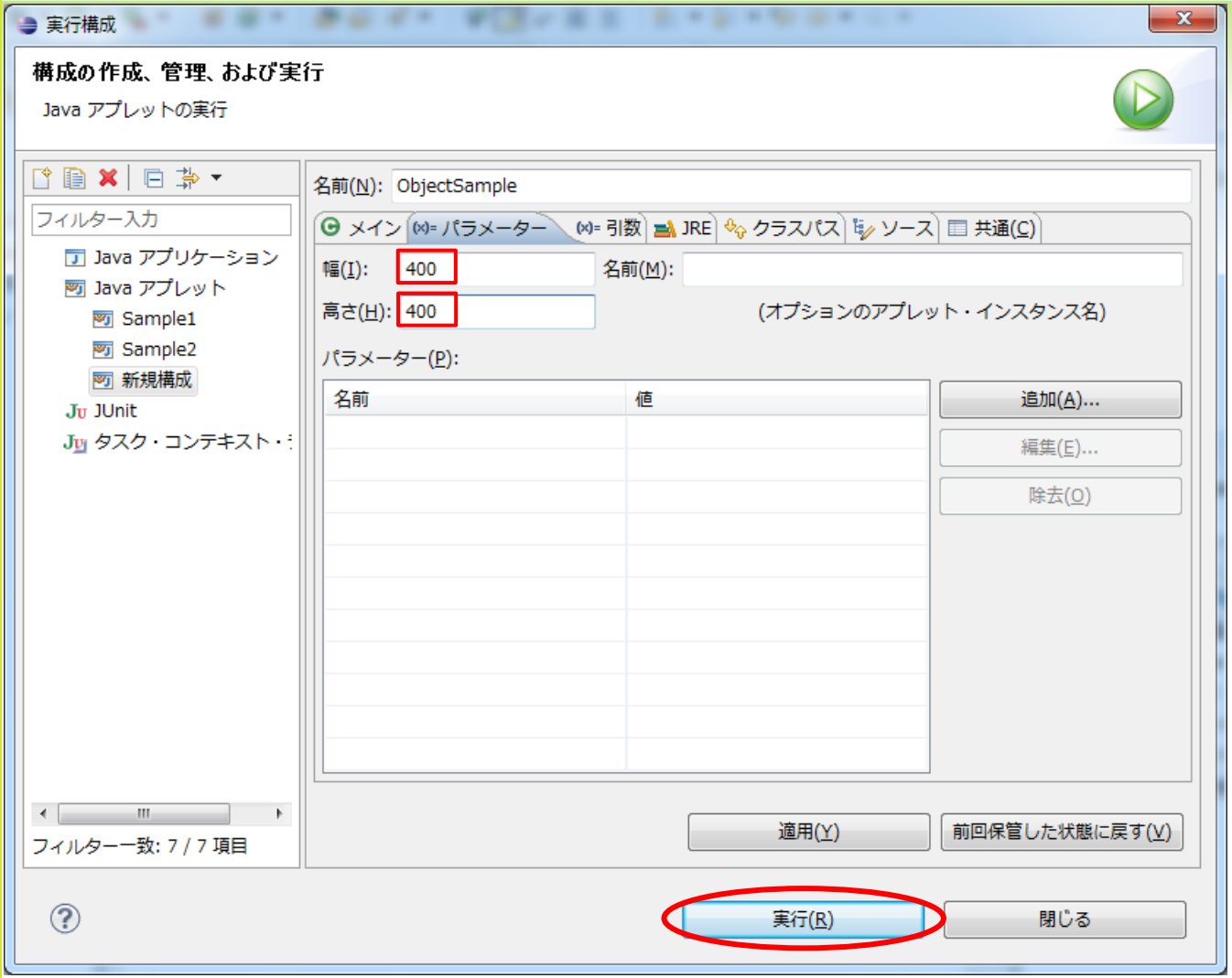
アプレットの選択
ObjectSampleを選択する



アプレットのクラス名を確認



アプレットの大きさを400x400に

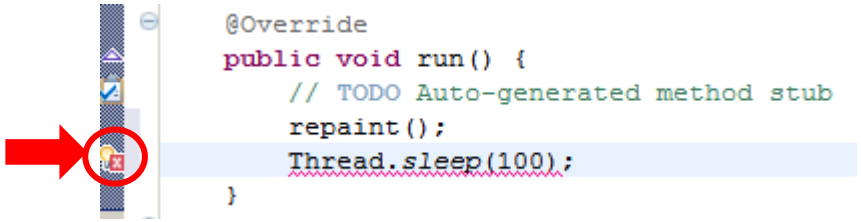


実行結果

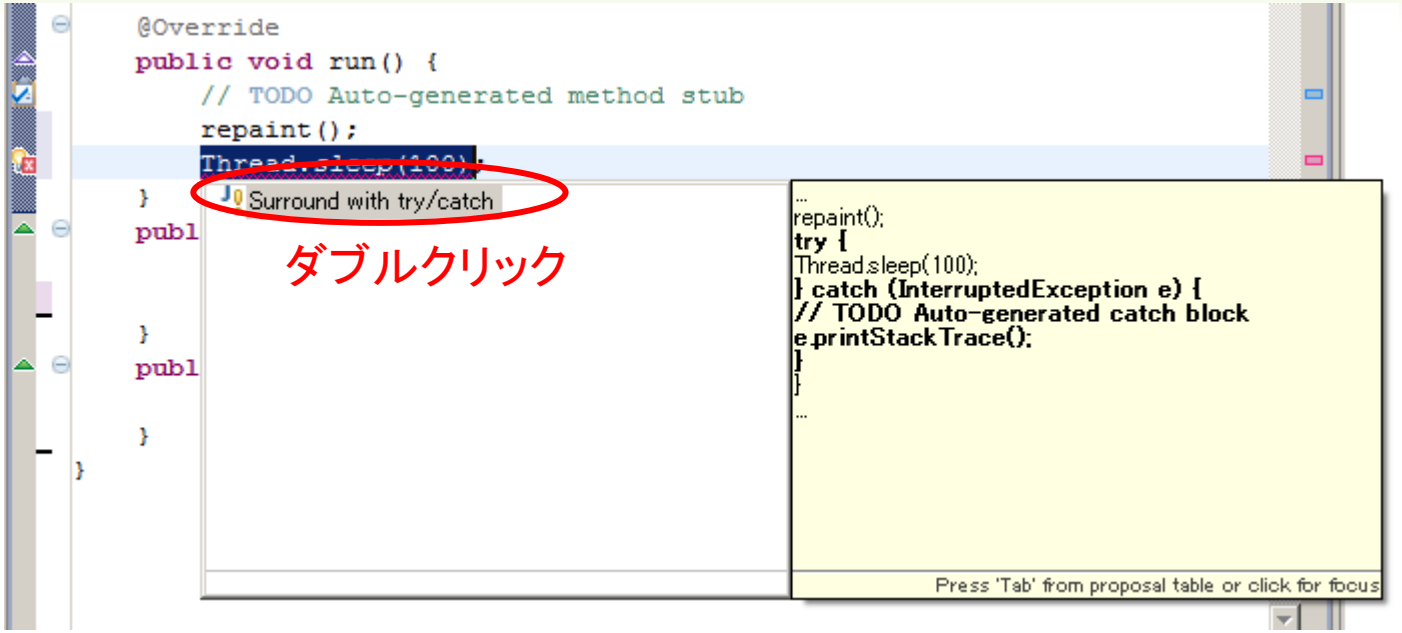
黒い丸が表示される位置はランダム

アニメーションするようにする

- アプレット本体では、repaint()で画面全体を書きなおして、100msecスリープする



SLEEPで例外(エラー)が発生するのでTRYで囲む



Tryが自動的に追加された

```
public void run() {
    // TODO Auto-generated method stub
    repaint();
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

REPAINT()をTRYの中に入れる

```
@Override
public void run() {
    // TODO Auto-generated method stub
        try {
            repaint();
            Thread.sleep(100);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}
```

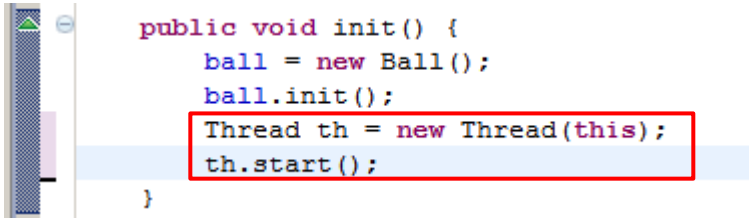
前の部分をWHILE(TRUE)で囲む

画面を書きなおす処理を無限に行う。
これがないとボールの移動が画面に反映されない

```
public void run() {
    // TODO Auto-generated method stub
        try {
            while(true) {
                repaint();
                Thread.sleep(100);
            }
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
}
```

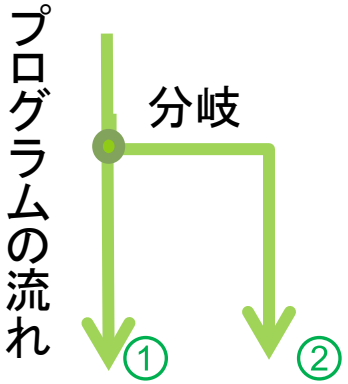
Threadをスタートさせる

- Threadとはプログラムの流れ。これを並行に進めるために、スレッドを分岐させる。
  Threadをstart()させると、run()が呼ばれる。



RUNNABLE

- プログラムの流れ（スレッド）を分岐して、それぞれ動かすことができる
- 基本的にはThreadクラスを使うが、Javaの場合にはインタフェースにRunnableを指定して、スレッドを生成することで、そのクラスのインスタンスの動作を、分岐させることができる



①では、ボタンなどのイベントを処理する流れ

②は新しく作った流れで、repaint()を繰り返す処理 run()を実行する

Ballもそれぞれが自分で動く流れを作る

Ball.javaを編集

```
import java.awt.Graphics;


public class Ball implements Runnable {
    int x,y;

    public void init() {
        x = (int)(Math.random()*400);
        y = (int)(Math.random()*400);
        new Thread(this).start();
    }

    public void paint(Graphics g) {
        g.fillOval(x, y, 5, 5);
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        try {
            while(true) {
                y = y + 5;
                Thread.sleep(100);
            }
        }catch (Exception e) {

        }
    }

}
```

ここでは、毎回yが5ずつ増える処理を繰り返す

sleepの量で早さが変わる

実行して動くことを確認

ボールの数を増やしてみよう

- まずボールを配列にする（ []を追加)

ObjectSample.javaを編集

```
public class ObjectSample extends Applet implements Runnable {
    Ball  ball;
```

```
public class ObjectSample extends Applet implements Runnable {
    Ball[]  ball;
```

他のところがエラーになる

```
public void init() {
    ball = new Ball();
    ball.init();
    Thread th = new Thread(this);
    th.start();
}
public void paint(Graphics g) {
    ball.paint(g);
}
```

入れ物を10個作る

```
ball = new Ball[10];
```

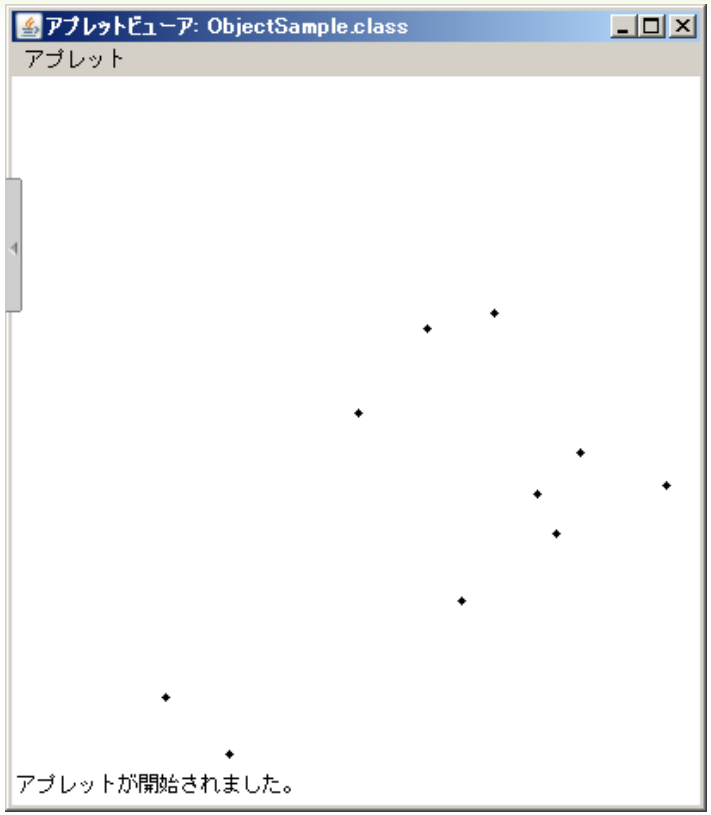
```
for(int i=0; i<ball.length; i++) {
    ball[i] = new Ball();
    ball[i].init();
}
```

1個ずつボールの実体を生成(new)して、それぞれを初期化(init())する

```
for(Ball b: ball) {
    b.paint(g);
}
```

拡張forでボールすべてをpaint()する

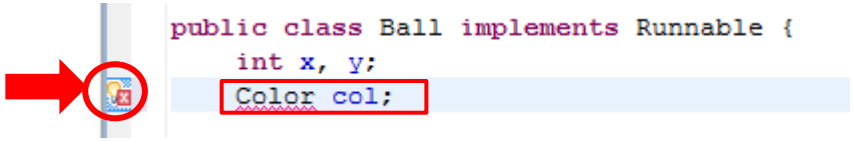
実行してみる



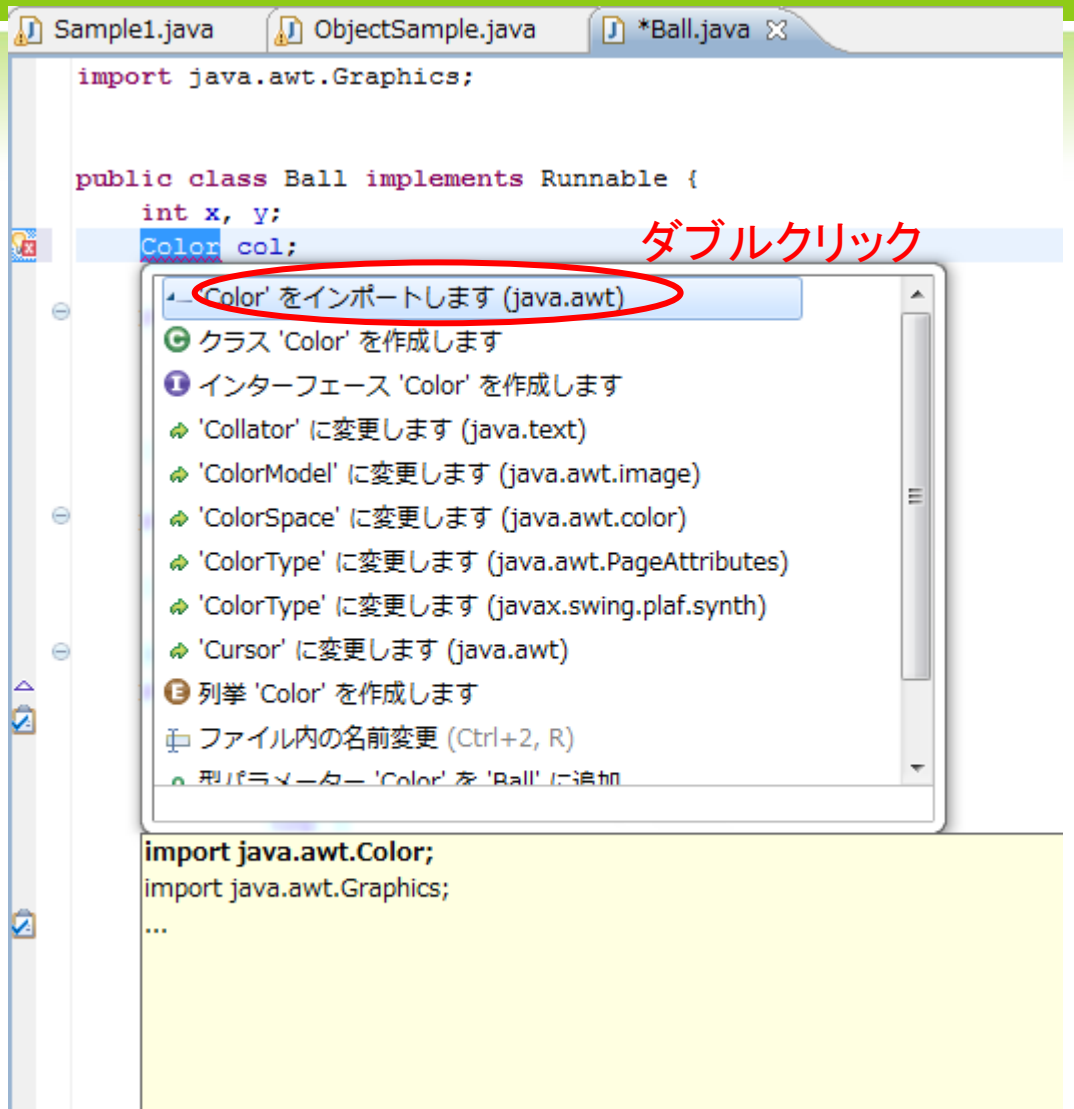
ボールに色をつける

- まずColorクラスを導入して、ボールの色をcol変数に持たせる

Ball.javaを編集



Colorをインポートスル



色の値は乱数で決める

Red, Green, Blueの3つの値(0～255)の値を設定する

```
public void init() {
    x = (int)(Math.random()*400);
    y = (int)(Math.random()*400);
    col = new Color(
            (int)(Math.random()*256),
            (int)(Math.random()*256),
            (int)(Math.random()*256)
            );
    new Thread(this).start();
}
```

色を付けるには、円を描く前に色をセットする

```
public void paint(Graphics g) {
    g.setColor(col);
    g.fillOval( x, y, 5, 5);
}
```

実行してみる

移動の仕方を関数にする

```
public void run() {
    // TODO Auto-generated method stub
    try {
        while(true) {
            y = y + 5;
            Thread.sleep(100);
        }
    }catch (Exception e) {

    }
}
```

```
public void move() {
    y = y + 5;
}
@Override
public void run() {
    // TODO Auto-generated metho
    try {
        while(true) {
            move();
            Thread.sleep(100);
        }
    }catch (Exception e) {

    }
}
```

呼び出す

課題

- init(), move()関数を編集して、個々のボールがランダムな方向に移動するようにせよ（ただし同じボールは途中で移動方向が変わってはいけない）
  - 基本課題：上下左右に移動する
  - 発展課題：ランダムに自由な角度で移動する
    - 移動する角度を変数で持つ (変数x,yなどと同じ)
    - Math.sin(), Math.cos()が使える(ラジアン)
    - πは、Math.PI
    - Math.toRadian(角度)でラジアンに変換できる
    - 移動量は整数　　（ (int)で整数化する ）

http://java.sun.com/javase/ja/6/docs/ja/api/java/lang/Math.html

課題の提出

- Z:¥workspace¥TestProject2¥src¥Ball.javaを添付ファイルとして提出すること。ファイル中にコメントして学籍番号が入っていること
- 件名：情報処理演習１０ 学籍番号　名前
- 宛先：kamahara@port.kobe-u.ac.jp