第3章　Javaのデータ型とクラス

この章では、Java に特有のデータ型・クラスに関する話題（多次元配列・総称クラス・ゴミ集めなど）をいくつか紹介する。また、Java のプログラムで頻繁に利用することになる重要なメソッドなどもここで紹介する。

例外処理の **try〜catch** 文は C 言語にはない構文なので、ここで紹介する。

Java のその他の制御構造の構文（**if** 文、**for** 文、**while** 文）は基本的には C と全く同じである。制御構造の復習を兼ねて、これらの制御構造を使った例題を取り上げる。

3.1　boolean 型

Java の if 文は C 言語と同じ書き方である。

if（条件式）文 $_1$
if（条件式）文 $_1$ **else 文** $_2$

条件式が成り立てば文 $_1$ を実行する。1 番めの形式は条件式が成り立たなければ何もしない 。2 番めの形式は文 $_2$ を実行する。 文 $_1$, 文 $_2$ は、当然ブロック（“{” と “}” で括った文の並び）でも良い。

ここで、条件式の型は _____型である。_____か _____の 2 つの値を取り得る型である。（boolean 型は既に紹介した Graphics クラスの draw3DRect や fill3DRect の引数としても用いられていた。） C 言語と異なり整数型（int 型）とは区別されている。このため（C 言語では OK だった） while (1) ... のような文はエラーとなる。

問 3.1.1 *int* 型と *boolean* 型を区別することの長短をまとめよ。

.....
.....
.....
.....
.....

条件判断文としてはこの他に **switch〜case 文**もあるが、C 言語と同じなので、ここでは説明を割愛する。

例題 3.1.2
時間の足し算を行なう。
ファイル *AddTime.java*

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

public class AddTime extends JApplet {
  int hour1, minute1, hour2, minute2;

  @Override
  public void init() {
    hour1   = Integer.parseInt(getParameter("Hour1"));
    minute1 = Integer.parseInt(getParameter("Minute1"));
    hour2   = Integer.parseInt(getParameter("Hour2"));
    minute2 = Integer.parseInt(getParameter("Minute2"));
  }

  @Override
  public void paint(Graphics g) {
    int hour, minute;
    // まず単純に足し算
    hour   = hour1+hour2;
    minute = minute1+minute2;

    if (minute>=60) {         // 繰り上がりの処理
      hour++;
      minute-=60;
    }
    // 結果を出力
    g.drawString("答えは "+hour+"時間 "+minute+"分です。", 30, 25);
  }
}
```

例えば、2 時間 45 分と 1 時間 25 分を足すと、そのままでは答が 3 時間 70 分になってしまう。分の部分が 60 以上になったときは繰り上げの処理を行なう処理を行なう必要がある。

3.2　文字列（String）に関する演算子とメソッド

Java では、________を用いて ____（あるいは、String 型と int 型のオブジェクトを String 型に変換したものを連接する）ことができる。

例:

```
System.out.println("2+2 は" + (2+2));
System.out.println("2+3 は" + (2+3) + "です。");
```

一方、JDK 5.0 からは C 言語のような書式指定を行う printf や sprintf メソッドに相当するメソッドも使用できる。上の drawString の場合、String.format というクラスメソッドを使って、次のように書くこともできる。

```
g.drawString(String.format("答えは %d 時間 %d 分です。", hour, minute), 30, 25);
```

詳細: この printf のようなメソッドは利用するのは簡単だが、総称クラス（Generics）・オートボクシング（Autoboxing）・可変個の引数（Varargs）など、いろいろな考え方が組合せられている。このうち総称クラスについては後述する。

可変個の引数を持つメソッドは API のドキュメントでは、

```
public static String format(String format, Object... args)
```

のように... を使って表されている。（この format メソッドは java.lang.String クラスのクラスメソッドである。）

_____は文字列から整数に変換するためのメソッド（java.lang.Integer クラスのクラスメソッド）である。

```
public static int parseInt(String s)
```

文字列同士が同じ文字列がどうかを判定するには String クラスの _____というメソッドを用いる。String クラスに対する==演算子は物理的に同じオブジェクトかどうかを判定するので、==の結果が true にならなくても、equals の結果が true になることがある。

java.lang.String クラス
  public boolean equals(Object s)
この文字列と指定されたオブジェクトを比較する。
  public boolean equalsIgnoreCase(String s)
この文字列と指定された文字列を比較する。大文字小文字を区別しない。

3.3 Java の例外処理

try〜catch〜文は

try ブロック$_1$ **catch** (例外型 変数) ブロック$_2$

という形で用いる。

この形は、まずブロック$_1$を実行する。ブロック$_1$の中で _____（エラーと考えて良い）と呼ばれる状況が起こったとき、catch の後ろの例外型がその例外の型と一致するか調べ、一致すればその後のブロック$_2$を実行する。一致するものがなければ、現在実行しているメソッドを呼び出した式を囲んでいる try〜catch 文を探す。それもなければ、さらにメソッド呼出しの履歴をさかのぼって、囲んでいる try〜catch 文を探す。それでもなければプログラムを終了する。

“**catch** (例外型 変数) ブロック” という形（catch 節）が複数続いても良い。その場合は最初に発生した例外にマッチする例外型を持つ catch 節が選択される。変数に初期値として、例外の情報をもつオブジェクトが渡されてブロックが実行される。

また、最後に “**finally** ブロック” という形（finally 節）がつく場合もある。その場合、finally ブロックは例外が起こったか否かにかかわらず、必ず実行される。

例えば、0 による除算を行なうと ArithmeticException という種類の例外が発生する。次のようなプログラムを実行すると、

ファイル TryCatchTest.java

```
public class TryCatchTest {
  public static void main(String[] args) {
    int i;
    for (i=-3; i<=3; i++) {
      try {
        System.out.println(10/i);
      } catch (ArithmeticException e) {
        System.out.println("エラー: "+e.toString());
      }
    }
  }
}
```

出力は次のようになる。

```
-3
-5
-10
エラー: java.lang.ArithmeticException: / by zero
10
5
3
```

i が 0 になった地点で例外が発生し、catch の後のブロックが実行される。その後は、プログラムの正常な実行を継続する。

ArithmeticException の他に、よく扱う必要のある例外としては次のようなものがある。いずれも java.lang パッケージに属する。java.lang パッケージは標準的なクラスを集めた、import の必要がない（最初から import されている）パッケージである。

NullPointerException	null が通常のオブジェクトとしてアクセスされた
NumberFormatException	Integer.parseInt などで 文字列が数値として解釈できない
ArrayIndexOutOfBoundsException	範囲外の添字で配列がアクセスされた

null は、＿＿＿＿＿＿＿＿＿＿として使われる定数である。（C 言語の NULL に対応する。）例えば、§ 2.5 で紹介した getParameter メソッドは、対応するパラメータがないときは null を返す。

例外の中には、発生する可能性があるときは try〜catch で囲んで処理する必要があるものもある。入出力に関する例外の java.io.IOException などである。

3.4　throw 文

プログラムにより例外を発生させるには throw 文を用いる。

throw 式;

この “式” は例外型（Exception あるいはそのサブクラス）のオブジェクトでなければならない。

次の例は、コマンドライン引数（main メソッドの引数の文字列の配列 args）として渡された数字の積を計算するプログラムである。途中で 0 が出てきた場合は、わざと例外を発生させて、残りのかけ算の処理を行なわないようにしている。（ただしこのプログラム例では、break 文を用いる方が自然である。）

ファイル TryCatchTest2.java

```
public class TryCatchTest2 {
  public static void main(String[] args) {
    int i, m=1;
    try {
      for (i=0; i<args.length; i++) {
        m *= foo(args[i]);
      }
    }  catch (Exception e) {
      m = 0;
    }
    System.out.println("答は " + m + "です。");
  }

  public static int foo(String arg) throws Exception {
    int a = Integer.parseInt(arg);
    if (a==0) throw new Exception("zero");
    return a;
  }
}
```

例えば “java TryCatchTest2 1 2 0 3 4 5 6” というコマンドライン引数で実行させると、3 番目の引数の 0 を呼んだ時点で、例外を発生させるため、残りの引数の 3, 4, 5, 6 は無視される。

上の foo メソッドのように本来 try〜catch で処理する必要のある例外を、メソッド内で処理しないメソッドは、throws というキーワードのあとに発生する可能性のある例外をコンマ（,）で区切って列挙しなければならない。

3.5 for 文, while 文

while (条件式 $_1$) 文 $_1$
for (式 $_1$; 式 $_2$; 式 $_3$) 文 $_1$
for (型 変数名 : 式) 文 $_1$

while 文は条件式 $_1$ が成り立つ間、文 $_1$ の実行を繰り返す。
1 つめの形式の **for** 文はループに入る前に、まず式 $_1$ を評価する。式 $_2$ が成り立つ間、文 $_1$、式 $_3$ の実行を繰り返す。2 つめの形式の **for** 文は JDK5.0 で導入されたものである。**for-each** 文と呼ばれることもある。（ただし、each というキーワードを使うわけではないので注意する。）この場合、式は直感的には何かの集まりを表すデータ型（配列など — 正確には配列またはインタフェース Iterable を実装するクラス）でなければならない。コロン（:）の前で宣言された変数に、この列の要素が順に代入され、文の実行が繰り返される。この形式の **for** 文の使用例はもう少し後で紹介する。

繰り返し文としてはこの他に **do〜while 文**もあるが、C 言語と同じなのでここでは説明を割愛する。

例題 3.5.1 正多角形の描画
整数 n をパラメータとして受け取り、正 n 角形を描画する。

ファイル N˙gon.java

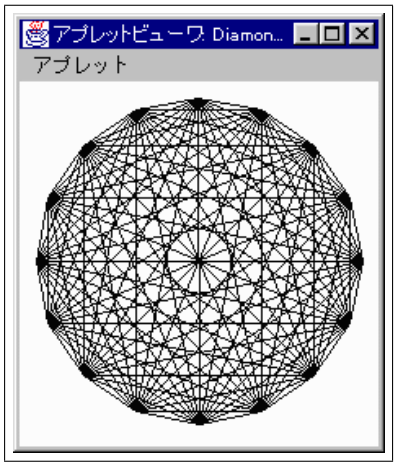
```
import javax.swing.*;
import java.awt.*;
import static java.lang.Math.*;

public class N_gon extends JApplet {
  int numPoints;
  int sc = 100;

  @Override
  public void init() {
    numPoints = Integer.parseInt(getParameter("NumPoints"));
  }

  @Override
  public void paint(Graphics g) {
    int i;
    double theta1, theta2;
    for(i=0; i<numPoints; i++) {
      // 単位 ラジアン
      theta1 = PI*2*i/numPoints;       // 360*i/n 度
      theta2 = PI*2*(i+1)/numPoints;  // 360*(i+1)/n 度
      g.drawLine((int)(sc*(1.1+cos(theta1))), (int)(sc*(1.1+sin(theta1))),
                 (int)(sc*(1.1+cos(theta2))), (int)(sc*(1.1+sin(theta2))));
    }
  }
}
```

`Math.PI` は 円周率 π（=3.1415 . . . ）、`Math.sin, Math.cos` は正弦、余弦関数である。これらはクラスフィールド、クラスメソッドである。

問 3.5.2 *sin, cos* などの数学関数のグラフを描くアプレットを書け。
参考: *(JDKDIR)/docs/ja/api/java.lang.Math.html*

問 3.5.3 正 *n* 角形のすべての頂点を結んでできる図形（ダイアモンドパターン）を描画するアプレットを書け。

問 3.5.4 色のグラデーション（*2* 次元 — 縦方向と横方向が別の色に変わる）を作成するアプレットを書け。



ダイアモンドパターン

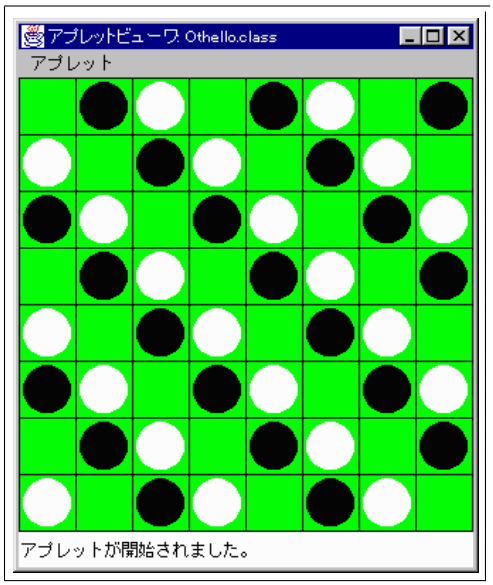
2 次元のグラデーション

（参考）1 次元のグラデーション

（参考） 1 次元のグラデーション

ファイル Gradation1.java

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

public class Gradation1 extends JApplet {
  int scale = 4;

  @Override
  public void paint(Graphics g) {
    int i;

    for (i=0; i<64; i++) {
      g.setColor(new Color(i*4, 0, 255-i*4));
      g.fillRect(i*scale, 0, scale, scale*10);
    }
  }
}
```



Othello.java

例題 3.5.5 グラフの描画

整数のデータを与え、そのデータの棒グラフを描く。

ファイル Graph.java

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

public class Graph extends JApplet {
  int[] is = {10, 4, 6, 2, 9, 1};
  Color[] cs = {Color.RED, Color.BLUE};
  int scale = 15;

  @Override
  public void paint(Graphics g) {
    int i, n = is.length;              // 配列の大きさ

    for (i=0; i<n; i++) {
      g.setColor(cs[i%2]);             // %は余り
      g.fillRect(0, i*scale, is[i]*scale, scale);
    }
  }
}
```

配列オブジェクトの _____というフィールド (?) によって配列の大きさ（要素数）を知ることができる。これも C 言語と異なる点である。**for** 文の中のブロックは変数 i が 0〜n-1 まで変化する間、繰り返される。

3.6　String クラスの split メソッド

例題 3.6.1 文字列の分割
数値の配列のデータを *HTML* の *param* タグを使って空白区切りの文字列で渡せるように、*Graph.java* を拡張する。

ファイル Graph.html

```
<html>
<head></head>
<body>
<applet code="Graph.class" width="200" height="200">
<param name="ARGS" value="10 4 6 2 9 1"> <!-- 数を空白で区切って渡す -->
</applet>
</body>
</html>
```

ファイル Graph.java

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

public class Graph extends JApplet {
  ...

  @Override
  public void init() {
    String[] args = getParameter("ARGS").split(" ");  // 区切りは空白
    int i;
    int n = args.length;
    is = new int[n];

    for(i=0; i<n; i++) {
      is[i] = Integer.parseInt(args[i]);              // 文字列を整数に変換
    }
  }
  ...
}
```

ここでは、空白区切りの文字列を文字列の配列に分割するために String クラスの _____メソッドを用いた。

java.lang.String クラス
  public String[] split(String regex)
この文字列を、指定された正規表現 (regex) に一致する位置で分割する。

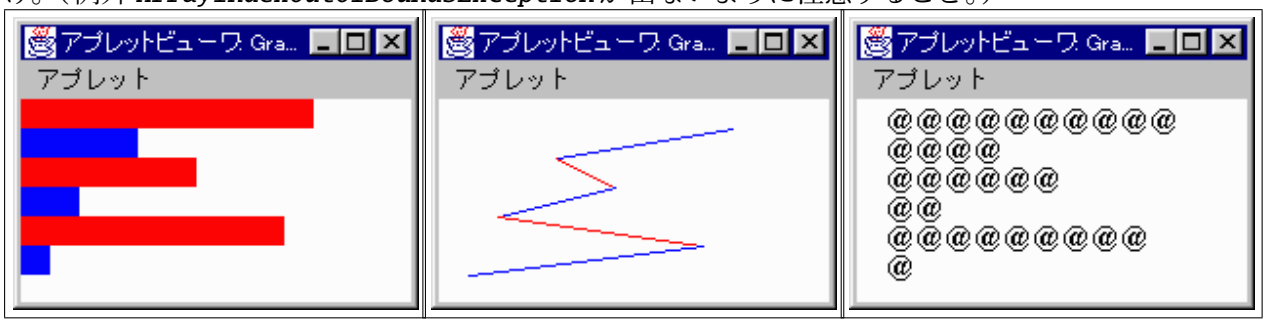
さらに Integer.parseInt メソッドで文字列から整数へ変換している。上の init メソッドの中身は、空白で区切られた文字列を配列に変換する典型的な方法である。split メソッドのの引数は、区切りに使用する文字列を表す正規表現である。これを","に変更すると、コンマで区切られた文字列を分割することができる。また、_____にすると、空白文字が 2 つ以上連続したり、タブ文字などが混ざったりという場合にも対応できる。Java で使用できる正規表現については、java.util.regex.Pattern クラスのドキュメント（*(JDKDIR)*/docs/ja/api/java/util/regex/Pattern.html）を参照すること。

3.7 配列の生成

new オペレータは配列を生成するときにも使用することができる。＿＿＿＿＿＿＿＿は、動的に長さ n の（int 型の）配列を生成する式である。int の代わりに他の型名を使うとその型の配列が生成される。C の配列宣言とは異なり、要素数 n の値がコンパイル時に定まっている必要はない。この形式を使うと配列の各要素は 0（オブジェクト型の場合は null）に初期化される。

一方、＿＿＿＿＿＿＿＿＿＿＿＿は、要素数を指定するのではなく、初期値を列挙して（int 型の）配列を生成する式である。

問 3.7.1 *HTML の param タグで与えられた数値データから折れ線グラフを生成するアプレットを書け。（例外 ArrayIndexOutOfBoundsException が出ないように注意すること。）*



棒グラフ　　折れ線グラフ　　キャラクターによるグラフ

（**注:** *n* 個の点を結ぶ線は *n-1* 本）

例題 3.7.2 時間のデータを *"9:45 12:35 4:42"* というように、空白で区切ってパラメータとして渡し、その時間の合計を表示するアプレットを書け。

ファイル AddTime2.java

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

public class AddTime2 extends JApplet {
  int[] t = {0,0};   // 初期値 0 時間 0 分

  int[] addTime(int[] t1, int[] t2) { // 時間の足し算を関数として定義する。
    int[] t3 = new int[2];   // 時間を大きさ 2 の配列で表す。

    t3[0] = t1[0]+t2[0];
    t3[1] = t1[1]+t2[1];
    if (t3[1]>=60) {  // 繰り上がりの処理
      t3[0]++;
      t3[1]-=60;
    }
    return t3;        // 新しい配列を返す。
  }

  @Override
  public void init() {
    String[] args=getParameter("Args").split("\\s+");

    for (String s : args) {
      String[] stime = s.split(":");
      int[] time = new int[] { Integer.parseInt(stime[0]),
                               Integer.parseInt(stime[1]) };
      t=addTime(t, time);
      // addTime 呼出し前に t と time に入っていた配列は不要となり、あとで GC される。
    }
  }

  @Override
  public void paint(Graphics g) {           // 結果を出力
    g.drawString("答えは "+t[0]+"時間 "+t[1]+"分です。", 30, 25);
  }
}
```

AddTime2.java では時間の足し算の処理はメソッド addTime として独立させた。paint や init のようなスーパークラスにあるメソッドの上書き（オーバーライド）ではなく、新しいメソッドの追加になる。このメソッドは戻り値を持つが、return 文の書き方も C 言語と同じである。

```
戻り値型 メソッド名（引数の型 引数名, ... ） {
  ...
}
```

というメソッドの定義の書き方も（戻り値型のまえに public などの修飾子がつくことがあることを除けば）C 言語の関数の書き方と同じである。

addTime はその中で配列を確保して戻り値に用いている。このように new は、C 言語の _____ に近い働きをする。また、このようにして確保された配列は、init の中で addTime を呼ぶときに次々と捨てられるが、これは _____（_____, GC）によって自動的に回収される。

（C 言語のように free による明示的なメモリの解放は必要ない。）GC のある言語ではこのように次々と新しいデータを生成して、古いデータを捨てるというスタイルが可能になる。

init メソッドは、拡張 **for** 文（**for-each** 文）を使用している。

3.8 多次元配列

例題 3.8.1 *int* 型の *8*× *8* の大きさの配列の配列を調べて、*1* なら白丸、*2* ならば黒丸を画面上の対応する位置に描画する。

ファイル Othello.java

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

public class Othello extends JApplet {
  int scale = 40;
  int space = 3;
  int[][] state =
        {{0,1,2,0,1,2,0,1}, {2,0,1,2,0,1,2,0}, {1,2,0,1,2,0,1,2},
         {0,1,2,0,1,2,0,1}, {2,0,1,2,0,1,2,0}, {1,2,0,1,2,0,1,2},
         {0,1,2,0,1,2,0,1}, {2,0,1,2,0,1,2,0}};

  @Override
  public void paint(Graphics g) {
    int i,j;

    for (i=0; i<8; i++) {
      for (j=0; j<8; j++) {
        g.setColor(Color.GREEN);
        g.fillRect(i*scale, j*scale, scale, scale);
        g.setColor(Color.BLACK);
        g.drawRect(i*scale, j*scale, scale, scale);
        if (state[i][j]==1) {
          g.setColor(Color.WHITE);
          g.fillOval(i*scale+space, j*scale+space,
                     scale-space*2, scale-space*2);
        } else if (state[i][j]==2) {
          g.setColor(Color.BLACK);
          g.fillOval(i*scale+space, j*scale+space,
                     scale-space*2, scale-space*2);
        }
      }
    }
  }
}
```

2 次元配列（配列の配列）を宣言するには、上のように [] を *2* つ重ねる（*3* 次元以上も同様）。*C* 言語の場合のように要素数を宣言する必要はない。（ただし、*C* 言語でも最初の次元の要素数は省略することができる。）*state* は配列の配列で、例えば、*state[0][1]* は、*0* 番めの配列 {0,1,2,0,1,2,0,1} の *1* 番めの数だから _ である。つまりこの位置（*0* 列めの *1* 行め）には白丸が描画される。

注意: なお、Java の 2 次元配列と C の 2 次元配列はメモリ上の配置の仕方が異なる。（もっとも Java でメモリ上の配置を意識する必要はほとんどない。）このため Java では C では許されない次のような 2 次元配列（異なるサイズの配列が混在している）

```
int[][] xss = {{1}, {1,2}, {1,2,3}};
```

も使用できる。

3.9　総称クラスの使用

総称クラス（generic class）は、型パラメータを持つクラスのことで、JDK5.0 から導入された。代表的な総称クラスの例として ArrayList, HashMap, LinkedList などがあげられる。型パラメータは＿＿＿＿＿＿書かれる。

ArrayList は＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿である。ArrayList の型パラメータは要素の型を表す。（総称クラスはこのようにコレクション（データの集まり）の型に使われることが多い。）例えば、String 型を要素とする ArrayList はArrayList<String>となり、次のように使用する。

```
ArrayList<String> arr1 = new ArrayList<String>();    // 空の ArrayList 作成
arr1.add("aaa");  arr1.add("bbb");  arr1.add("ccc"); // データ追加
String s = arr1.get(1);                               // データ取出し
```

add メソッドでデータを追加し、get メソッドでデータを取り出すことができる。

int, double のようなプリミティブ型は総称クラスの型パラメータになることができないという制限があるので注意が必要である。このときは Integer, Double などの対応する＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿と呼ばれるクラスを利用する。ラッパークラスとプリミティブ型の変換はほとんどの場合、自動的に行われる（オートボクシング）ので、int の代りに Integer と書く以外は通常のクラス型をパラメータとするときと変わらない。例えば次のように書くことができる。

```
ArrayList<Integer> arr2 = new ArrayList<Integer>();  // 空の ArrayList 作成
arr2.add(123);  arr2.add(456);  arr2.add(789);       // データ追加
int i = arr2.get(1);                                 // データ取出し
```

ArrayList<String>に int 型の要素を add したり、ArrayList<Integer>から String 型の要素を get したりするのは、当然型エラー（コンパイル時のエラー）になる。

```
ArrayList<String> arr1 = new ArrayList<String> ();
arr1.add(333);          // 型エラー

ArrayList<Integer> arr2 = new ArrayList<Integer> ();
...
String t = arr2.get(2)  // 型エラー
```

このような型エラーをコンパイル時にちゃんと発見したい、というのが、総称クラスの導入のそもそもの動機である。

API ドキュメントの中では、型パラメータは E のような仮のクラス名が使われ、

java.util.ArrayList<E>クラス:

```
public boolean add(E e)
```

リストの最後に、指定された要素（e）を追加する。

```
public E get(int index)
```

リスト内の指定された位置（index）にある要素を返す。

のように書かれる。

例題 3.9.1 木の再帰的な描画

描画データの一時保存に ArrayList を使用する例である。init メソッドで描画データを保存し、paint メソッドでそれを利用している。なお、配列型も総称クラスの型パラメータとして問題なく使用することができる。この例の場合、描画データの要素数が前もって（簡単には）わからないので、配列ではなく ArrayList を使用している。

また、この例では paint メソッドの中で、拡張 **for** 文（**for-each** 文）を使用している。

ファイル Tree.java

```
import java.awt.*;
import javax.swing.*;
import java.util.ArrayList;
import static java.lang.Math.*;

public class Tree extends JApplet {
  ArrayList<int[]> data = new ArrayList<int[]>();

  public void drawTree(int d, double x, double y, double r, double t) {
    /* d  --- 再帰呼出しの深さ     (x, y)  --- 枝の根元の座標        *
     * r  --- 枝の長さ              t       --- 枝の角度（ラジアン） */
    double r1;
    if (d==0) return;
    data.add(new int[] {(int)x, (int)y, (int)(x+r*cos(t)), (int)(y+r*sin(t))});
    r1 = r;      drawTree(d-1, x+r1*cos(t), y+r1*sin(t), 0.5*r, t+0.2);
    r1 = 0.55*r; drawTree(d-1, x+r1*cos(t), y+r1*sin(t), 0.5*r, t+1.25);
    r1 = 0.45*r; drawTree(d-1, x+r1*cos(t), y+r1*sin(t), 0.5*r, t-1.3);
  }

  @Override
  public void init() {
    drawTree(6, 128, 255, 128, -PI/2);
  }

  @Override
  public void paint(Graphics g) {
    g.setColor(Color.GREEN);
    for(int[] pts : data) {       // for-each 文
      g.drawLine(pts[0], pts[1], pts[2], pts[3]);
    }
  }
}
```

例題 3.9.2 色の名前

HashMap は＿＿＿＿＿＿と呼ばれるデータ構造である。通常の配列と異なり、int 型だけではなく、任意の型（String 型など）をキー（添字）として、値を格納・検索することができる。HashMap の型パラメータは 2 つあり、1 つめがキーの型、2 つめが値の型である。下の例では、HashMap<String, Color>、つまりキーが String 型で値が Color 型の連想配列を用いている。値の格納には put メソッド、検索には get メソッドを用いる。

java.util.HashMap<K,V>クラス:

public V put(K key, V value)

指定された値（value）と指定されたキー（key）をこのマップに関連付ける

public V get(Object key)

指定されたキー（key）がマップされている値を返す。

Object（java.lang.Object）クラスは Java のすべてのクラスのスーパークラスとなる、クラス階層のルートクラスである。

ファイル ColorName.java

```
import java.awt.*;
import javax.swing.JApplet;
import java.util.HashMap;

public class ColorName extends JApplet {
  HashMap<String, Color> hm;
  String color1, color2, color3;

  @Override
  public void init() {
    //  和色大辞典 http://www.colordic.org/w/ より
    hm = new HashMap<String, Color>();
    hm.put("赤", new Color(0xed1941)); hm.put("黄", new Color(0xffd400));
    hm.put("緑", new Color(0x45b97c)); hm.put("青", new Color(0x009ad6));
    hm.put("紫", new Color(0x8552a1)); ...

    color1 = getParameter("Color1");
    color2 = getParameter("Color2");
    color3 = getParameter("Color3");
  }

  @Override
  public void paint(Graphics g) {
    g.setFont(new Font("Sans", Font.BOLD, 64));
    g.setColor(hm.get(color1)); g.drawString(color1, 10, 70);
    g.setColor(hm.get(color2)); g.drawString(color2, 90, 70);
    g.setColor(hm.get(color3)); g.drawString(color3, 170, 70);
  }
}
```

問 3.9.3 総称クラス *LinkedList* の使用法を調べ、プログラムを作成せよ。

キーワード　if 文, if〜else 文, boolean 型, Integer.parseInt メソッド, while 文, for 文, for-each 文, 配列, length メソッド, split メソッド, static, Math クラス, 多次元配列, 総称クラス, ArrayList クラス, HashMap クラス, LinkedList クラス