1　参照はオブジェクトの分身

変数やメソッドの集合体であるオブジェクトはサイズが大きく、そのままの形で代入などの操作に使用すると多くの処理時間がかかります。そこで、Java 言語では、処理時間を短縮できるようにオブジェクトを直接操作せず、その分身を通して処理することにしています。それが「参照」です。プログラムでは参照をオブジェクトとみなして操作します。参照はオブジェクトへのリンク情報を持っているので、参照に対する操作はそのままオブジェクトに対する操作になるからです。この章を修了すると参照の意義と機能を理解し、正しく使用できるようになります。

内　容

＜準備＞　ライブラリの準備（mylib パッケージ）

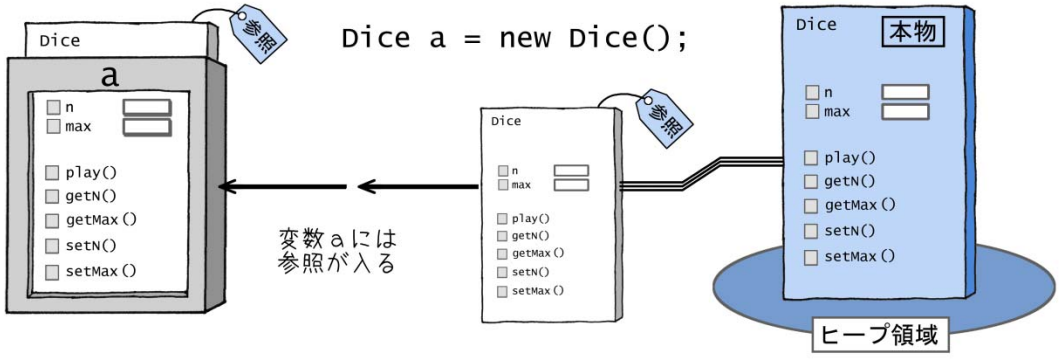
1．オブジェクトは参照で操作する

2．配列はオブジェクトの仲間

3．null はリンク情報をもたない参照

✧ この章のまとめ

✧ 通過テスト



― オブジェクトを作ると変数にはその参照が入る ―

※この章を学習するに当たり、Eclipse でプロジェクト「Part2_1」を作成してください。

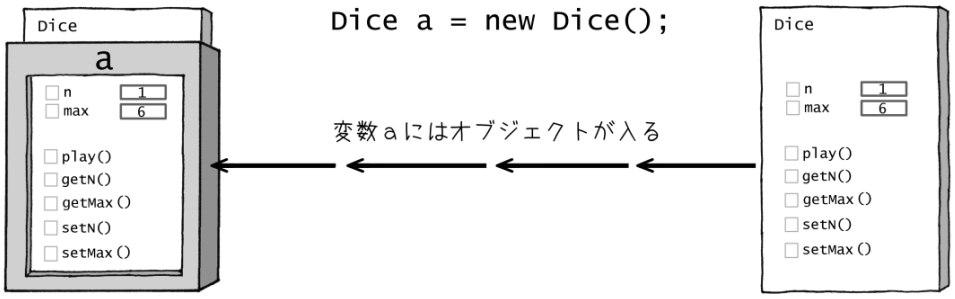
ライブラリの準備（mylib パッケージ）

学習に必要なクラスをパッケージに入れて用意しています。パッケージはサポートウェブ（巻末資料参照）からダウンロードできます。

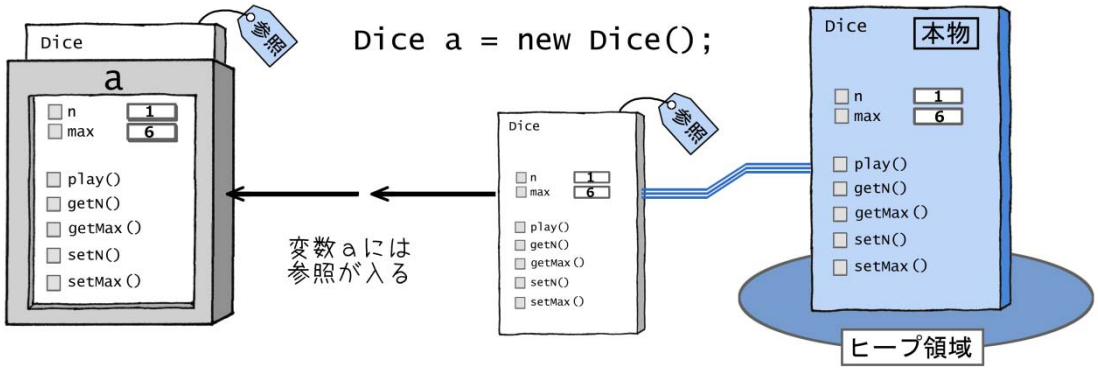
巻末の「資料１ mylib パッケージの読み込み手順」を見て、mylib パッケージを Part2_1 プロジェクトに読み込んでください。例題などでは mylib パッケージのクラスをインポートして使います。mylib パッケージの内容は「資料２　mylib パッケージのクラス」に解説がありますが、この章の本文でも解説しています。

1.オブジェクトは参照で操作する

これまでの説明では、オブジェクトは new 演算子で作成され、それが変数に代入されるという理解でした。例えば、Dice a = new Dice();を実行したときは、次の図のように、変数 a にオブジェクトが格納されているというイメージです。



しかし、実際は少し違います。実は、作成したオブジェクトの実体はメモリ上のヒープ領域と呼ばれる場所に置かれます。下の図で「本物」と書かれている青いオブジェクトがそれで、これがオブジェクトの唯一の実体です。



変数に代入されるのは「参照」というタグの付いた白いオブジェクトで、これは本物ではありません。しかし、白いオブジェクトは、青いオブジェクトの分身として働きます。というのも、この白いオブジェクトに対するすべての操作が、そのまま、本物である青いオブジェクトに伝えられるからです。

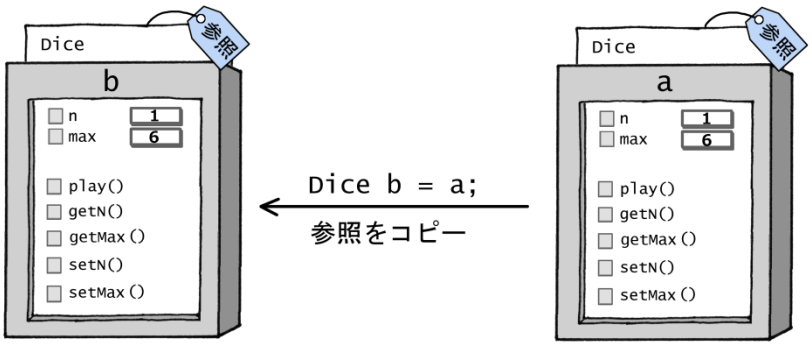
つまり、白いオブジェクトと青いオブジェクトは常にリンクしているのです。白いオブジェクトを使って何かを行うと、それらはすべて、本物の青いオブジェクトでの操作になります。この白いオブジェクトを、青いオブジェクトの参照と呼びます。

■ 参照を使う理由　ー参照は素早くコピーできて効率がよいー

変数に本物のオブジェクトを入れない理由は、サイズが大きいため、コピーに時間がかかるからです。プログラムは値のコピーで成り立っています。実際、代入文での代入処理、メソッドの仮引数と実引数の間での値の受け渡し、メソッドの戻り値の受け取りなど、すべて値のコピーです。これらで本物を使っていると全体の効率が落ちてしまうのです。

これに対して、参照（白いオブジェクト）は、コンピュータの内部では本物の青いオブジェクトへのリンク情報にすぎません。実体ではないのでサイズが小さく、コピーが短い時間で済みます。本物を使うよりはるかに効率のよい処理が可能なのです。

```
Dice b = a;  // a から b へ、参照のコピーは効率のよい処理である
```



■ 参照型には３種類ある

代入やメソッドでの値の受け渡しが素早くできるという理由で、オブジェクトの代わりに参照を使います。参照は型の分類では参照型です。Java 言語の型は次のように、参照型と基本データ型からなり、参照型にはクラス型以外にも配列型やインタフェース型（Part3-4 で解説）が属しています。

そこで、参照を入れる変数を参照型変数といいます。参照型変数は、クラス型、配列型、インタフェース型のどれであっても、中にオブジェクトの参照を格納する点では同じです。

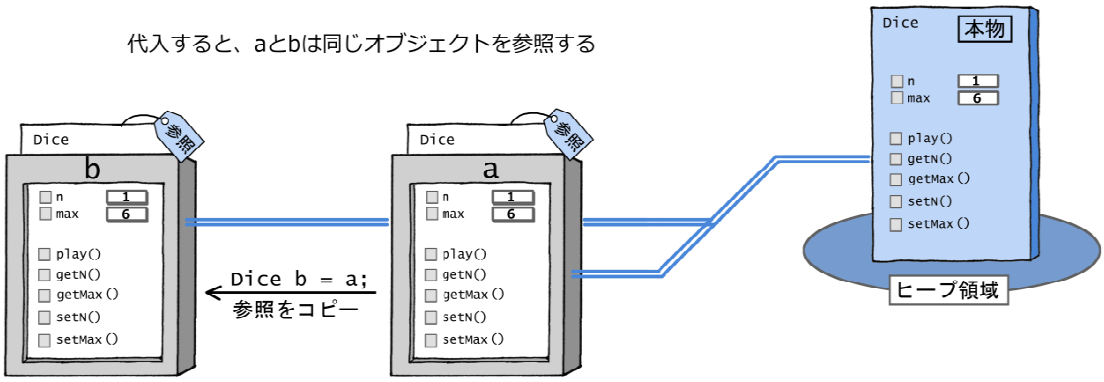
【Java 言語のデータ型】

参 照 型	基本データ型
クラス型	整数型
インタフェース型	浮動小数点型
配列型	文字型
	論理型

■ 参照型変数の代入ではオブジェクトではなく参照がコピーされる

参照型の変数の代入は、参照がコピーされるだけで、本物のオブジェクトはコピーされないことに注意してください。参照型の変数同士で代入を行うと、次の図のように両方の変数が同じ参照を持ち、同じオブジェクトにリンクします。メソッドの引数や戻り値として参照を受け渡す時も、実質的に代入が行われるので同じことが起こります。

これを同じオブジェクトを参照するといいます。



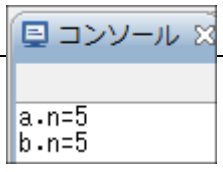
次の例題はこの点を確認します。

例題１　参照型の変数の代入では参照がコピーされる

Exec.java

```
1   package sample1;
2   import mylib.Dice;  // mylib パッケージから Dice クラスを import する
3   public class Exec {
4       public static void main(String[] args) {
5           Dice  a, b;
6           a  =   new Dice();    // オブジェクトを作成して a に（参照を）代入する
7           b  =   a;             // b に a を代入（a の参照がコピーされる）
8           a.play();             // a の目数を変更する
9           System.out.println("a.n="+a.getN());// a の目数を表示する
10          System.out.println("b.n="+b.getN());// b の目数を表示する
11      }
12  }
```

実行結果 ▷

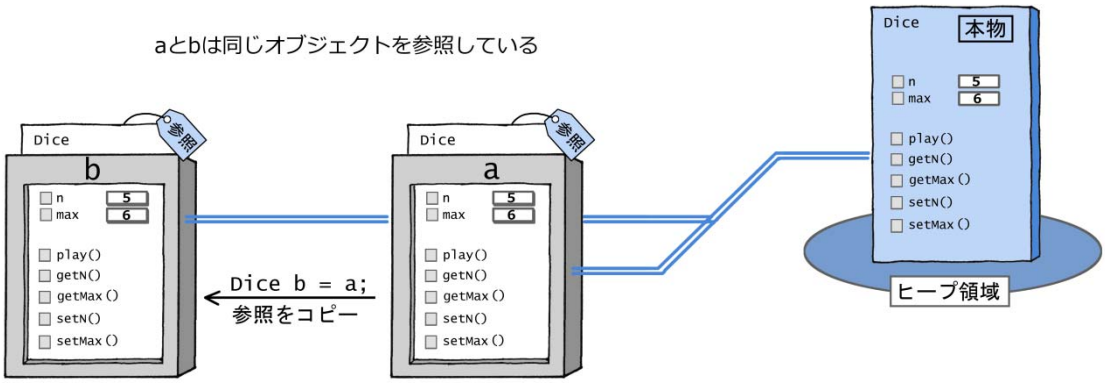


【解説】

2 行目の import 文は mylib パッケージにある Dice クラスを取り込みます。この Dice クラスは 3 章の例題 2-1 とほぼ同じものです（この後のコラムにメソッドなどの一覧表を掲載しています）。

さて、例題の Exec クラスでは、Dice クラスの変数 a、b が宣言されています。最初にオブジェクトを作成して a に代入し、次に、b=a;とします（5～7 行目）。

b=a;では、a の"参照"が b にコピーされることに注意してください。これで、a も b も同じ参照をもつようになり、その結果、同じオブジェクトにリンクします。



これを確認するため、8 行目で a.play()を実行して、a の目数（フィールド変数 n の値）をランダムに変更します。そして、9、10 行目で a と b の目数を表示し、両方が同じ値であることを確認しています。

結果はどちらも 5 で、同じオブジェクトを参照していることがわかりました（疑う人は何度か実行してみて下さい）。

参照型変数

・参照型変数には参照が入っている
・参照は本物のオブジェクトの分身であり、常に本物のオブジェクトにリンクしている
・参照型の変数同士の代入では、オブジェクトではなく参照がコピーされる
・参照型の変数同士の代入では、両方の変数が同じオブジェクトを参照するようになる

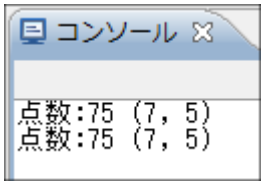
■ Dice クラスの API

クラスの使い方の一覧表を示します。クラスの使い方がわかるこのような情報を API（Application Programming Interface）といいます。ソースコードは mylib パッケージの中にあるので、Eclipse で開くと見ることができます。

Dice クラス　(Dice.java)	
■フィールド	
private int　n	サイコロの目数
private int　max	目数の最大値
■コンストラクタ	
public Dice(int n, int max)	目数、最大値を引数にとる
public Dice(int max)	Dice(1, max)と同じ。n=1 とみなす
public Dice()	Dice(1, 6) と同じ。n=1, max=6 とみなす
■メソッド	
public int play()	n の値を 1～max の間でランダムに変更する
public int getN()	n の値を返す
public int getMax()	max の値を返す
public void setN(int m)	n の値を m にする
public void setMax(int m)	max の値を m にする

練習 1

1.次のようなプログラム（Exec1.java）を作成しなさい



△ 実行結果

◆import --- mylib パッケージから ChipSet クラスをインポートする

main()

- 準備 ― ChipSet クラスの変数 a, b を宣言する
- 処理 ― ChipSet クラスの引数のないコンストラクタを使って a にオブジェクトを代入する
  a = new ChipSet();
  - b = a と代入する
  - a のチップを 75 点分だけ増やす　a.change(75)
  - a のチップ数を表示する （ChipSet クラスの toString()メソッドを利用する）
    → System.out.println(a) とする
  - b のチップ数を表示する （同上）

（注）ChipSet クラスの API を示します。青い部分がここで使用するコンストラクタとメソッドです。

ChipSet クラス （ChipSet.java）	
■フィールド	
private int　c10	10 点のチップの数
private int　c1	1 点のチップの数
■コンストラクタ	
public ChipSet(int c10, int c1)	引数により c10 と c1 の値をフィールドにセットする
public ChipSet(int p)	点数 p を c10 と c1 の数に変換してセットする
public ChipSet()	引数は(0,0)とみなす（c10 も c1 も 0 にする）
■メソッド	
// チップ枚数を加算する public int change(int c10, int c1)	引数 c10,c1（正または負の値）をフィールド変数に加算し、加算後の枚数を点数に換算した値を返す
// チップ点数を枚数に変換して加算する public int change(int p)	点数 p（正または負の値）をチップ枚数に換算して c10、c1 に加算する。また、加算後の点数を返す
// チップ枚数を点数に変換した値を返す public int getPoints()	チップ枚数を点数に換算した値（= c10×10+c1）を返す
// オブジェクトを表す文字列を返す public String toString()	c10、c1 の合計点数と、c10、c1 の値を表す文字列を返す 例えば、c10 が 9, c1 が 2 なら、 "点数:92 (9,2)" を返す

2．次のプログラムを実行するとき、正しいものはどれか記号で答えなさい。

```
package ex1;
import mylib.Dice;
public class Exec2 {
    public static void main(String[] args) {
        Dice  a, b, c;          // クラスの内容は、「Dice クラスの API」を参照
        a   = new Dice(1,52);   // 目数を 1、最大値を 52 にしてオブジェクトを作成する
        c   = b   =   a;
        c.setN(25);
        b.setN(13);
        System.out.println(a.getN());
    }
}
```

《選択肢》

ア　 1　と表示する
イ　25　と表示する
ウ　13　と表示する
エ　コンパイルエラーになる
オ　実行時例外 (注) が起こる

（注）実行時例外とは、実行した時にコンソールにエラーが表示され停止してしまう状態です。文法的な間違いのコンパイルエラーとは違います。この後の例題 3 で実例を解説しています。

解答＿＿＿＿＿＿＿＿

3．次のプログラムを実行する時、正しいものはどれか記号で答えなさい。

```
package ex1;
import mylib.Dice;
public class Exec3 {
    public static void main(String[] args) {
        Dice  a, b;           // クラスの内容は、「Dice クラスの API」を参照
        a = new Dice(1, 52); // 目数を 1、最大値を 52 にしてオブジェクトを作成する
        b = new Dice(a.getN(), a.getMax());
        b.setN(10);
        System.out.println(a.getN()+"/"+b.getN());
    }
}
```

《選択肢》

ア　1/1　と表示する
イ　1/10　と表示する
ウ　10/10　と表示する
エ　コンパイルエラーになる
オ　実行時例外が起こる

解答＿＿＿＿＿＿＿＿

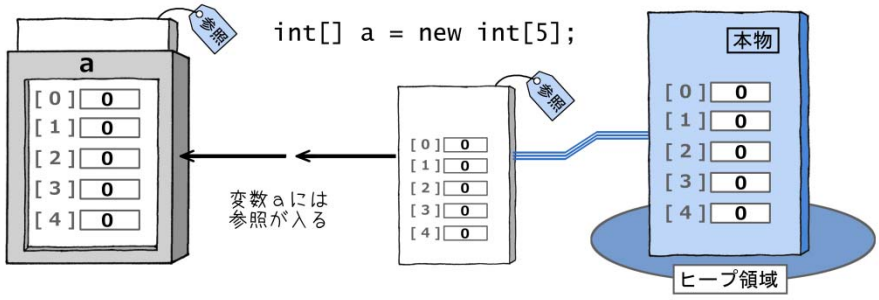
2．配列はオブジェクトの仲間

次は要素を 5 つ持つ配列を new を使って作成する式ですが、本来、new はヒープ領域にオブジェクトを作成する演算子です。このことからも配列がオブジェクトの仲間であることが分かります。

```
int[] a = new int[5];       // 作成した配列はヒープ領域に置かれる
```

配列もサイズが大きいので本体はヒープ領域に置かれます。そして、次の図のように、変数 a には本体ではなく配列の参照が代入されます。

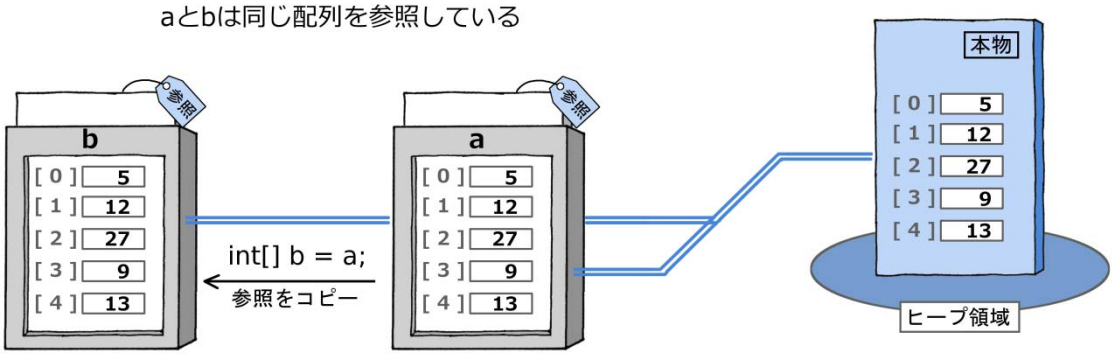
オブジェクトと同様に、参照に対するすべての操作は、本物の配列に伝達され、配列自身の操作になります。つまり、参照と、本物の配列は常にリンクしているのです。



また、配列変数を、他の変数に代入したりメソッドでの値の受け渡しに使うと、変数の中身は参照ですから、参照がコピーされるだけです。配列本体はコピーされません。

例えば次の図は、{5,12,27,9,13}を値に持つ配列 a を、b に代入する操作です。b には a の内容である参照がコピーされ、両方の変数が同じ配列にリンクされます。そのため、2 つの変数は同じ配列を参照することになります。

```
int[] a = {5, 12, 27, 9, 13};    // 配列を作成
int[] b = a;                     // b に a を代入する
```



次の例題は、このことを確かめます。

例題 2　配列変数の代入では参照がコピーされる

Exec.java

```
1   package sample2;
2   public class Exec {
3       public static void main(String[] args) {
4           int[] a   = {5, 12, 27, 9, 13};// 配列を作成
5           int[] b   = a;                 // a を b に代入
6           b[2]      = 0;                 // b[2]を 0 にする
7           for(int n : a){
8             System.out.print(n+" ");     // a の要素を表示する
9           }
10          System.out.println();          // 改行
11          for(int n : b){
12            System.out.print(n+" ");     // b の要素を表示する
13          }
14      }
15  }
```

実行結果 ▷

【解説】

配列 a は、初期値{5, 12, 27, 9, 13}を指定して作成しています。これは次のように new で配列を作成してから、各要素に値を代入する処理と同じです。便利な書き方ですが、配列本体がヒープ領域に作成されることに変わりありません。

```
int[] a = new int[5];
a[0] = 5;
.....  省略 .....
a[4] = 13;
```

5 行目で配列 a を配列 b に代入しています。これにより、配列本体ではなく、変数 a の中にある参照が、変数 b にコピーされます。その結果、a と b は同じ参照を持ち、同じ配列本体を参照するようになります。

a と b がリンクする配列本体は同じものなので、6 行目で b[2]=0;としていますが、a[2]=0;と書いても同じです。a を使っても b を使っても結果は同じなわけです。

7～13 行では拡張 for 文を使って、a と b の要素を表示しています。実行結果のように表示される値が全く同じであり、a と b が同じ配列を参照していることが確かめられます。

なお、メソッドの引数や戻り値として配列を受け渡す時も、実質的に代入が行われるので、同じことが起こります。次の練習問題でそれを確かめてください。

練習 2

1.次のプログラムを実行したとき、正しいものを選んで記号で答えなさい。

```
1   package ex2;
2   public class Exec1 {
3       public static void main(String[] args) {
4           int[] a   =   new int[5];
5           sub(a);
6           for(int n : a){
7             System.out.print(n);
8           }
9       }
10      public    static void sub(int[] b){
11          for(int i=0; i<b.length; i++){
12            b[i] = i;
13          }
14      }
15  }
```

《選択肢》

ア 00000　と表示する
イ 01234　と表示する
ウ 何も表示しない
エ コンパイルエラー

解答＿＿＿＿＿＿

2．次のプログラムを実行したとき、正しいものを選んで記号で答えなさい（やや難しい問題）。

```
1   package ex2;
2   public class Exec2 {
3       public static void main(String[] args) {
4           int[] a   =   new int[5];
5           sub(a);
6           for(int n : a){
7             System.out.print(n);
8           }
9       }
10      public    static void sub(int[] b){
11          int[] c = {0, 1, 2, 3, 4 };
12          b = c;
13      }
14  }
```

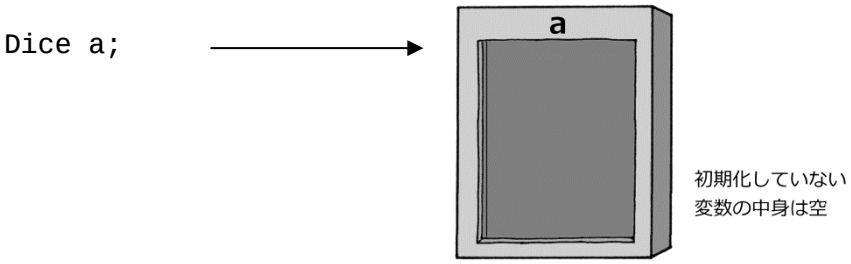
《選択肢》

ア 00000　と表示する
イ 01234　と表示する
ウ 何も表示しない
エ コンパイルエラー

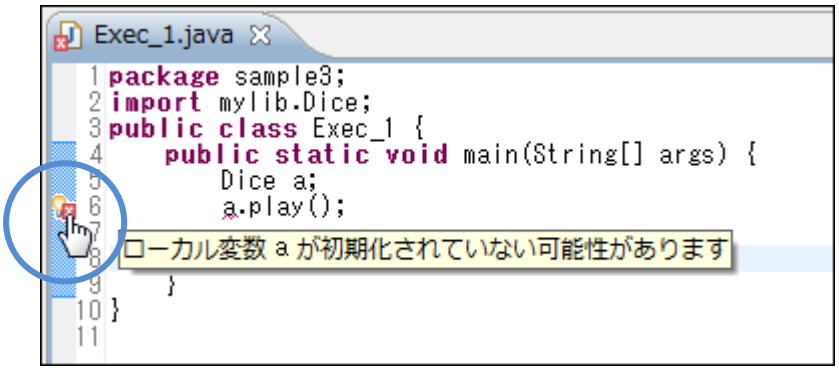
解答＿＿＿＿＿＿

3.null はリンク情報をもたない参照

Java 言語では、宣言しただけの変数は空です。これは参照型でも同じで、文字通り何も値が入っていない状態です。この状態を初期化されていないといいます。

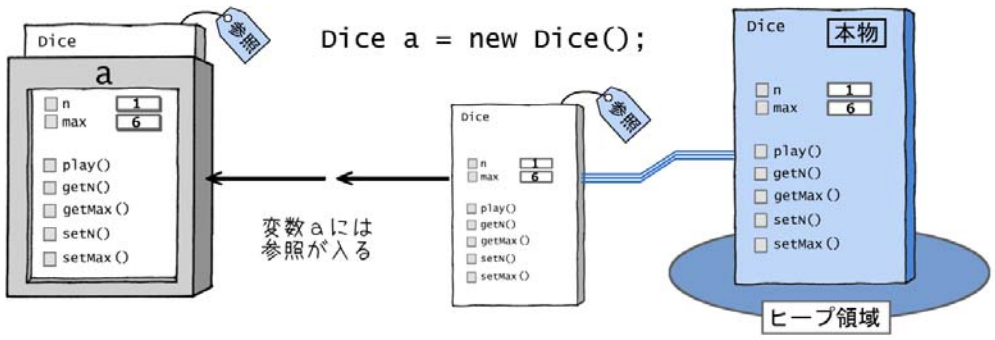


初期化されていない変数を演算に使用するとコンパイルエラーになります。Eclipse 上では、次のように使用した部分に赤いエラーマークが表示されます。

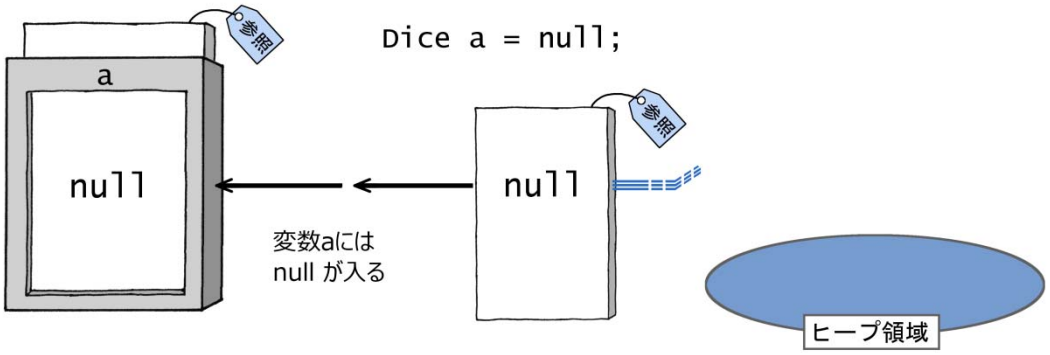


参照型の変数を初期化するには、new 演算子でオブジェクトを作成して（その参照を）代入します。

```
Dice a = new Dice();   // オブジェクトを代入して初期化する
```



また、null（ヌル）を代入して、初期化の代わりとすることができます。ただし、null は、オブジェクトにリンクしていない参照です。

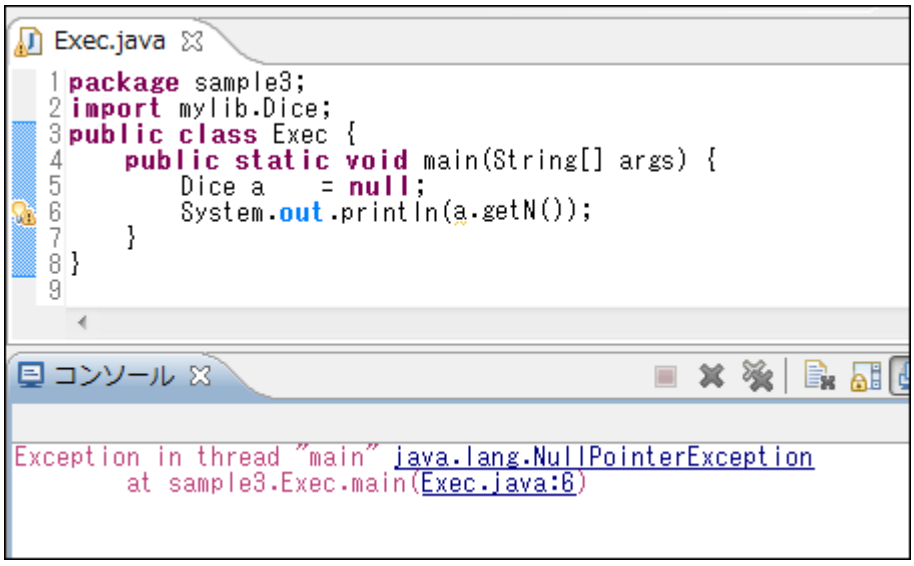


ここで、大事なことは null の入った参照型変数は、コンパイラからは初期化されているとみなされることです。ですから、ある変数が null で初期化されたのか、オブジェクトにリンクしている参照で初期化されたのかはプログラマが判断しなければなりません。

判断を誤って、null が入っている参照型変数を使ってしまうと、実行時例外が発生します。十分注意してください。次に Eclipse の画面を使って実行時例外の例を示します。

例題 3　実行時例外

Exec.java



【解説】

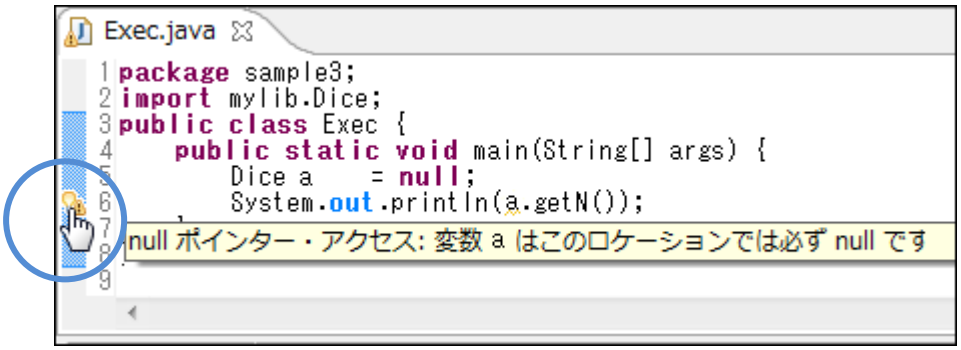
例題は 5 行目で Dice 型の変数 a に null を代入し、初期化の代わりにしています。そして、6 行目で、a.getN()の返す値をコンソールに表示します。Eclipse 上のプログラムを見ると、どこにも赤いエラーマークが付いておらず、プログラムにはコンパイルエラーがないことがわかります。

しかし、実行するとコンソールにエラーが発生したというメッセージが表示され、プログラムは停止します。これが実行時例外です。コンソールの(Exec.java:6)という表示は6行目が原因であることを示しています。

6行目は、内容がnullであるaを使って、a.getN()を実行しています。nullはオブジェクトにリンクしていない参照なので、a.getN()を実行できません。これが実行時例外を発生した原因です。

■ Eclipseでの警告表示

なお、Eclipseでは、コンパイルエラーにはならないものの、行番号の左に警告マークが表示されます。その上にマウスポインタを乗せてみると以下のようなメッセージが表示され、aがnullであることが示されています。



練習 3

1.次のプログラムを実行するとき、正しいものはどれか記号で答えなさい。

(注）Dice クラスは mylib パッケージからインポートしています。

問 1

```
1   package ex3;
2   import mylib.Dice;
3   public class Exec1 {
4       public static void main(String[] args) {
5           Dice a=new Dice(1, 10), b;
6           a = b;
7           System.out.println(a.getN());
8       }
9   }
```

《選択肢》
ア 1 と表示する
イ 何も表示されない
ウ 5 行目でコンパイルエラー
エ 6 行目でコンパイルエラー
オ 7 行目でコンパイルエラー
カ 実行時例外が起こる

解答＿＿＿＿＿＿

問 2

```
1   package ex3;
2   import mylib.Dice;
3   public class Exec2 {
4       public static void main(String[] args) {
5           Dice a=new Dice(1,10), b=null;
6           Dice c = a = b;
7           System.out.println(c.getN());
8       }
9   }
```

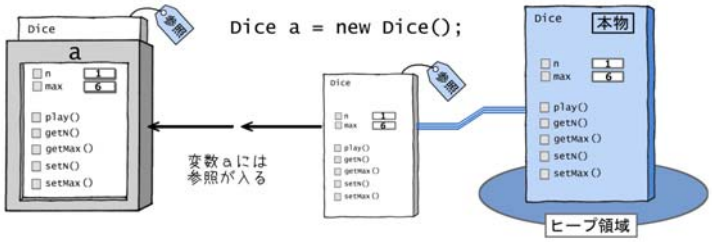
《選択肢》
ア 1 と表示する
イ 何も表示されない
ウ 5 行目でコンパイルエラー
エ 6 行目でコンパイルエラー
オ 7 行目でコンパイルエラー
カ 実行時例外が起こる

解答＿＿＿＿＿＿

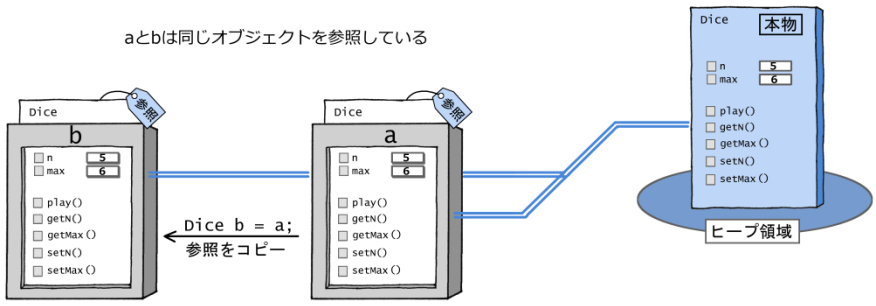
まとめ

オブジェクトは参照で操作する

□new 演算子で作成したオブジェクトはヒープ領域に置かれる
□変数に代入されるのは本物のオブジェクトではなく参照である
□参照は常に本物のオブジェクトにリンクしている分身である
□参照に対する操作はすべて本物のオブジェクトに伝達され、オブジェクトで実行される
□参照は本物のオブジェクトに比べ、サイズが小さく効率のよい処理ができる

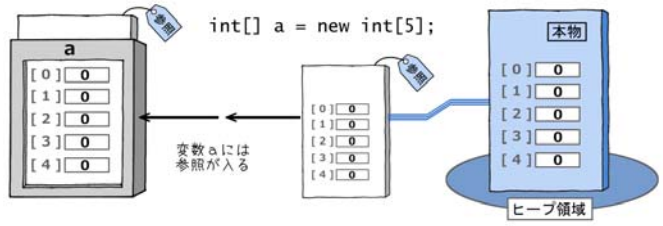


□参照を入れる変数の型を参照型という
□参照型にはクラス型と配列型、インタフェース型がある
□参照型の変数同士で代入を行うと、本物のオブジェクトではなく参照がコピーされる
□同じ参照を持つ変数は同じオブジェクトを参照する

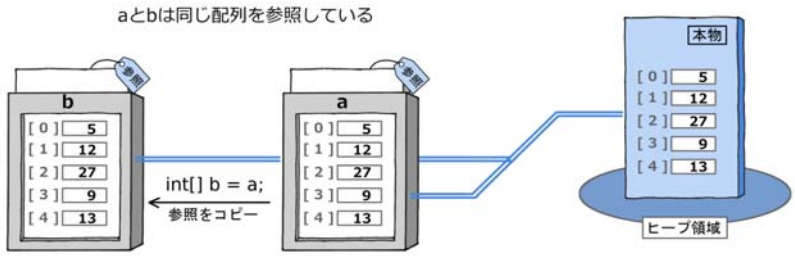


配列はオブジェクトの仲間

□配列を new で作成すると、配列本体はヒープ領域に置かれる
□配列変数に代入されるのは本物の配列ではなく参照である
□配列変数同士の代入では、本物の配列ではなく参照がコピーされる
□参照は常に本物のオブジェクトにリンクしている分身である

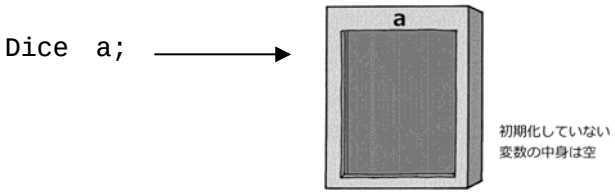


□同じ参照を持つ配列は同じ配列を参照する



null はリンク情報を持たない参照

□宣言しただけの変数は値がなく完全な空である

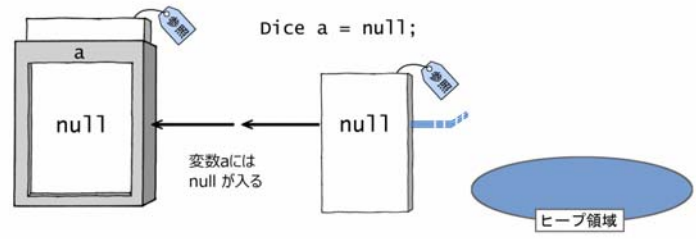


□空で値を持たない変数は初期化されていない変数という

□初期化されていない変数を演算に使うとコンパイルエラーになる

□参照型変数はオブジェクトを new で作成して（参照を）代入することにより初期化する

□null はオブジェクトにリンクしていない参照である



□参照型変数はオブジェクトの参照の代わりに、null を代入して初期化できる

□null を代入した変数を演算に使ってもコンパイルエラーにはならない

□ただし、null を代入した変数を演算に使うと、実行した時に実行時例外が発生する

通過テスト

1.

Dice a = new Dice(); として Dice オブジェクトを new で作成すると、オブジェクトは［　ア　］に置かれ、a には参照が入る。参照はオブジェクトではないが、オブジェクトの分身である。参照は［　イ　］ので、参照に対する操作はオブジェクトに対する操作と同じである。この意味で、a はオブジェクトを［　ウ　］するという。

また、Dice b = a; とすると、参照が a から b へコピーされ、a と b は［　エ　］を参照する。このとき、a に対する操作は b に対する操作と［　オ　］。例えば、フィールド変数 n に対するセッターである setN メソッドを使って、a.setN(7)とした後では、b のゲッターである b.getN()は［　カ　］を返す。

ア　①スタック領域　②ヒープ領域　③オブジェクト領域　④コード領域
イ　①オブジェクトな　②オブジェクトではない　③オブジェクトにリンクする
ウ　①参照する　②代行する　③代替する　④制御する
エ　①同じ参照　②同じオブジェクト　③異なる参照　④異なるオブジェクト
オ　①同じである　②同じではない　③ほぼ同じである
カ　①0　②1　③7　④5　⑤a による setN()以前の値

【解答】

ア	イ	ウ	エ	オ	カ

2.次の文ので正しいものに○、間違っているものに×を付けなさい。

(1)クラス型のオブジェクトを新規に作成するには new 演算子を使う以外ない
(2)参照型変数 a の値を同型の変数 b に代入するとオブジェクトをコピーできる
(3)配列もオブジェクトの一種なので、多くの配列変数はクラス型の参照型変数に代入できる
(4)初期化されていない配列変数 a を同型の変数 b に代入すると実行時例外が起こる
(5)配列変数は null で初期化できる
(6)null を代入した参照型変数 a は、同型の他の変数 b に代入できる
(7)null は、規定値の空白のオブジェクトを参照している
(8)オブジェクトを new で作成するとフィールドは規定値で埋められる

【解答】

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)

3.次のプログラムを実行するとき、正しいものはどれか記号で答えなさい。

ただし、Dice クラスのコンストラクタについては●頁の API を参照しなさい。

問 1

```
1   package pass2;
2   import mylib.Dice;
3   public class Exec {
4       public static void main(String[] args) {
5         Dice   a   =   new Dice(5,12);
6         Dice   b   =   new Dice(a.getN(), a.getMax());
7         a = b;
8         b.setN(10);
9         System.out.println(a.getN());
10      }
11  }
```

《選択肢》

ア　12 と表示する
イ　10 と表示する
ウ　 5 と表示する
エ　コンパイルエラー
オ　実行時例外

【解答】＿＿＿＿＿＿＿＿

問 2

```
1   package pass2;
2   public class Exec2 {
3       public static void main(String[] args) {
4         int[]      n   = {1, 2, 3, 4, 5};
5         double[]   x   = n;
6         for(double a : x){
7             System.out.print(a);
8         }
9       }
10  }
```

《選択肢》

ア　12345 と表示する
イ　1.02.03.04.05.0　と表示する
ウ　何も表示しない
エ　コンパイルエラー
オ　実行時例外

【解答】＿＿＿＿＿＿＿＿