プログラミング D Java 演習課題レポート

担当教員：肥後 芳樹, 小南 大智
提出者:09B22002 安西 俊輔
メール：u262691h@ecs.osaka-u.ac.jp

提出日:2023 年 12 月 24 日

1　作成したプログラムの操作方法と機能

1.1　操作方法

以下に、このライフゲームの操作方法を示す。
プログラム起動後、以下のように 12x12 マスの盤面とその下に 3 つのボタン [Next][Undo][New] が表示される。

盤面の中のセルを、マウスでクリックまたはドラッグすることで生死状態を切り替える。

”Next” ボタンをクリックすると、ライフゲームが 1 世代更新される。”Undo”ボタンをクリックすると、ライフゲームが 1 世代巻き戻る。”New”ボタンをクリックするとウィンドウが起動し、新たにシミュレーションを開始する。また、ウィンドウのサイズを変更すると、それに伴って盤面のサイズも変わる。それぞれの機能の詳細は、次節で説明する。

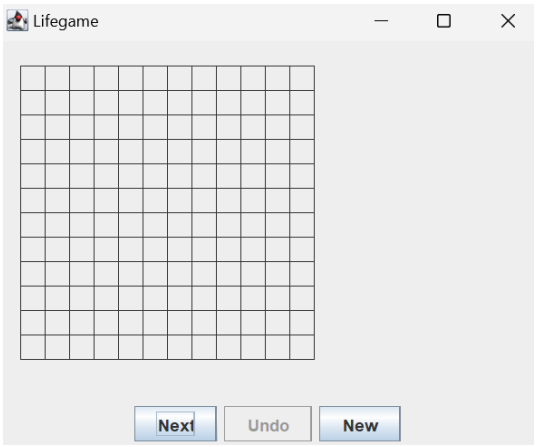
1.2　機能

以下に、このライフゲームのプログラムの機能を示す。

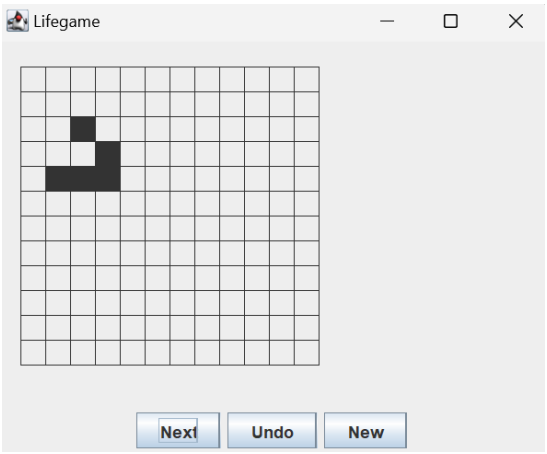
- 初期画面:
  - プログラムを起動すると、初期画面が表示される。
  - この初期状態では、12x12 のセルからなる盤面が表示され、事前に設定された生命体が存在する。そのセルが生きている状態だとそのセルが黒色で表示される。初めは、すべてのセルが死んでいる状態、つまり全てのセルが色なしの状態で表示される。
- クリックによるセルの生死の切り替え:
  - マウスを使用して、セルの生死を切り替えることができ、それに伴って表示も替わる。
- ドラッグによるセルの生死の切り替え:
  - マウスをドラッグすることで、複数のセルの生死をまとめて変更できる。
  - 同セル内でドラッグしても生死は切り替わらない。異なるセルに移動したときに移動先のセルの生死が切り替わる。
- Next ボタン:
  - ”Next” ボタンをクリックすると、ライフゲームが 1 ステップ進む。
  - セルの生死がルールに基づいて更新され、新しい状態が表示される。
- Undo ボタン:
  - ”Undo” ボタンは初めは無効になっている。一度”Next” ボタンを押す、もしくは一度セルの状態を切り替えた後に有効になり、前の状態に巻き戻すことができる。
  - 巻き戻りの回数には制限があり、最大 32 回まで巻き戻りが可能で、それ以上はボタンが無効になる。
  - セルの生死状態の切り替えは、1 マスごとに巻き戻りする。例えば、マウスのドラッグによるセルの切り替えが 5 個分行われた場合は、1 つずつセルの状態が巻き戻り、5 回”Undo”ボタンを押すと元の状態に戻る。
- New ボタン:

  - ”New” ボタンをクリックすると、新しいウィンドウが開き、新しいライフゲームのシミュレーションが開始される。
  - 新しく始まったシミュレーションは、同様に全ての機能が備わっている。
- ウィンドウサイズの操作:
  - ウィンドウサイズを変更したとき、縦幅と横幅の短い方に合わせて盤面のサイズも変わる。最小サイズには制限がある。
- ウィンドウタイトルバーへのプログラム名の表示
  - ウィンドウのタイトルバーに、「Lifegame」という文字列を表示する。

次ページで、実際にプログラムを動作したときの機能を図で説明する。

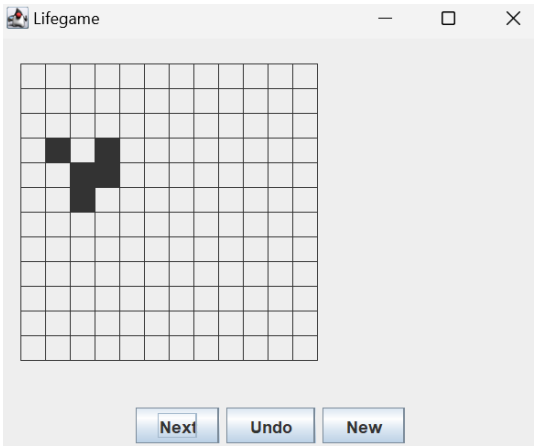


(a) 初期状態

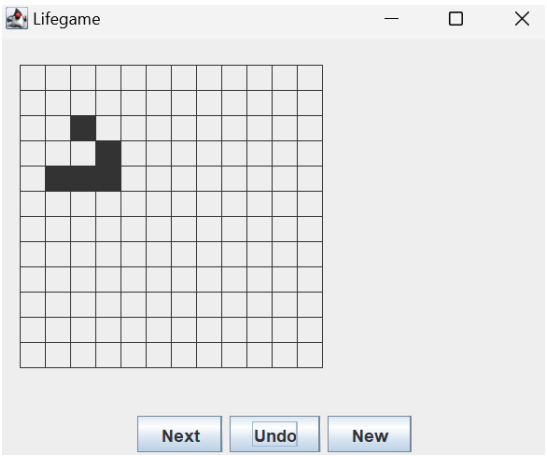


(b) 入力例

図 1: 初期状態とクリック例



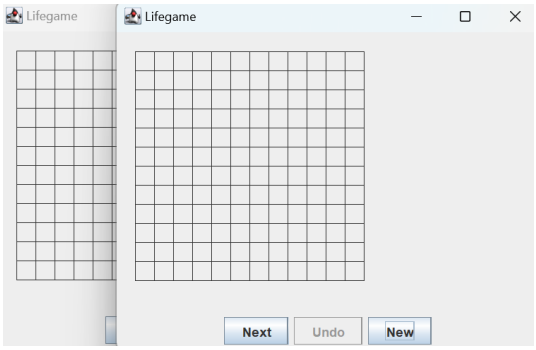
(a) 図 1(b) の状態で”Next”を 1 回クリックした



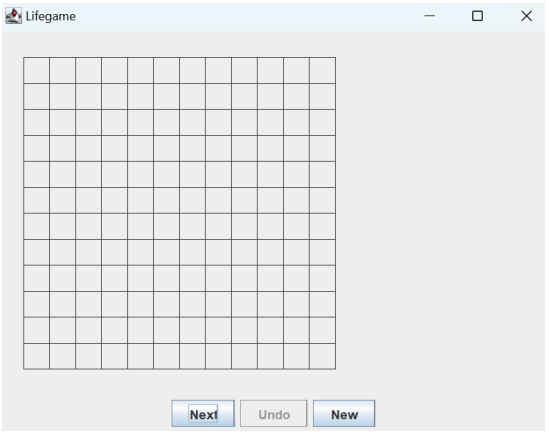
(b) その後”Undo”を 1 度クリックした

図 2: ”Next”と”Undo”の動作

プログラムを起動すると、図 1(a) のようなウィンドウがでてくる。すべてのマスが死んだ状態で、”Undo”ボタンが無効であることが確認できる。図 1(b) はセルの状態変化をした例である。クリック・ドラッグのどちらでもセルの生死状態を切り替えることができる。
図 2 では、図 1(b) の状態から”Next”ボタンと”Undo”ボタンを押したときの様子である。”Next”ボタンを押したとき、正常にライフゲームが 1 世代分更新されていることが分かる。その後、”Undo”ボタンを押すと、図 1(b) の状態に戻っている。



(a) ”New”を 1 度クリックした



(b) ウィンドウのサイズを大きくした

図 3: ”New”とウィンドウのサイズ変更

”New”ボタンをクリックすると図 3(a) のようになる。ウィンドウがもう 1 つ起動され、初期状態と同じ動作をする。ウィンドウのサイズを変更したら図 3(b) のようになる。ウィンドウサイズを変えたときも、全てのセルがウィンドウ内に収まって、セルの形を正方形に維持できている。

2　プログラムによる実現

2.1　クラスと機能の対応

以下に、プログラムを構成する各クラスがどの機能の実現を担当しているかを示す対応表を記述する。

クラス	機能・役割
BoardModel	• 盤面に関する機能を担当している。 • ユーザーのセルの生死状態の変更、世代の更新・巻き戻し、巻き戻し可能であるかの判断、指定された座標を含むセルの生死状態の確認など、ライフゲームのロジックを実装。 • BoardListener インターフェースを実装し、リスナーを通じて他のクラスに状態変更を通知。
BoardView	• BoardModel で取り扱う盤面の出力や入力に関する機能を担当している。 • マウスイベントを受け取り、BoardModel の変更をトリガーとする。 • JPanel を拡張しており、paintComponent メソッドをオーバーライドして盤面の描画を行う。
Main	• グラフィカルユーザーインターフェース（GUI）を提供し、BoardModel と BoardView を統合し、プログラムを実行する機能を担当している。 • Next、Undo、New ボタンのアクションリスナーを実装し、それぞれ BoardModel の適切なメソッドを呼び出す。
ModelPrinter	• デバッグ情報の出力。 • BoardModel の更新通知を受け取り、デバッグ情報をコンソールに表示。
BoardListener	• BoardModel の状態更新を通知するためのインターフェース。 • ModelPrinter が実装。

2.2 役割分担の方針と達成度

役割分担としては、クラス名に関するものについての記述を書くように分けた。例えば、BoardModel は、セルの状態変更などのライフゲームの盤面について取り扱う記述を、BoardView は、マウスのクリックなど入出力に関する記述を含むようにした。この方針は、かなり達成できていると思う。この後詳しく説明するが、BoardModel にある、座標がどこのセルに含まれているかを判定するメソッドは、BoardView に記述した方がコンパクトにできたが、この方針のためにそのように記述した。

3 各機能の実現方法

以下に、各機能の実現方法の詳細を記述する。

3.1 盤面の状態の更新に連動して更新される画面の表示項目と更新方法

BoardModel クラス内で盤面の状態が更新されたとき、fireUpdate メソッドが呼び出される。fireUpdate メソッドは BoardListener を通じて BoardView に通知され、repaint メソッドが呼び出されて画面が再描画される。更新される表示項目は、ボタンの有効・無効状態、盤面の生死状態である。

```
    @Override
public void paint(Graphics g) {
super.paint(g);

            //盤面の描画の記述

}
```

この paint メソッドは、JPanel クラスのメソッドをオーバーライドしている。この paint メソッドが呼び出されるタイミングは、repaint メソッドが呼ばれたときやウィンドウが再描画される際に呼び出される。BoardModel の状態が変更されると、その変更が BoardView クラスに通知され、fireUpdate メソッドを介して再描画が促される。また、ボタンの状態は setEnabled メソッドを用いて更新される。

次に、盤面の世代の更新時について説明する。以下は該当メソッドを抜粋したものである。

```
    public void next() {
        boolean[][] newCells = new boolean[rows][cols];
        for (int x = 0; x < cols; x++) {
            for (int y = 0; y < rows; y++) {
                int liveNeighbors = countLiveNeighbors(x, y);
                if (cells[x][y]) {
                    if (liveNeighbors < 2 || liveNeighbors > 3) {
                        newCells[x][y] = false;
```

```
                } else {
                    newCells[x][y] = true;
                }
            } else {
                if (liveNeighbors == 3) {
                    newCells[x][y] = true;
                } else {
                    newCells[x][y] = false;
                }
            }
        }
    }
    history.push(cells);
    if (history.size() > maxHistorySize) {
        history.pollLast();
    }
    cells = newCells;

    fireUpdate();
}
```

boolean[][] newCells = new boolean[rows][cols]; で、新しい状態の盤面を格納する配列 newCells を作成する。次に、二重の for ループを使用して、各セルの新しい状態を計算する。int liveNeighbors = countLiveNeighbors(x, y); で、セルの周囲の生存している隣接セルの数を取得する。countLiveNeighbors メソッドは以下である。

```
private int countLiveNeighbors(int x, int y) {
    int liveNeighbors = 0;
    for (int dx = -1; dx <= 1; dx++) {
        for (int dy = -1; dy <= 1; dy++) {
            if (dx == 0 && dy == 0) {
                continue;
            }

            int newX = x + dx;
            int newY = y + dy;

            if (newX >= 0 && newX < cols && newY >= 0 && newY < rows && cells[newX][newY]) {
                liveNeighbors++;
            }
```

```
            }
        }
        return liveNeighbors;
    }
```

このメソッドは、周囲の 8 マスのうち生きている状態のセルを数える。このメソッドで得られた値に基づいて生死状態の切り替えを行う。現状の盤面の記録を行った後に、計算した盤面を反映させたら、変更通知を送る。以上の仕組みで世代の更新を行う。

3.2 巻き戻しのための盤面の状態の記録方法

巻き戻しのための盤面の状態の記録方法は、Deque boolean[][] history というデータ構造を使用している。Deque は双方向キュー（double-ended queue）で、リストの両端で要素の追加や削除ができる。この Deque は boolean[][] 型の盤面状態を格納し、履歴を管理している。

以下に、具体的な操作方法とデータ構造の解説を示す。

```
    private Deque<boolean[][]> history;
    private int maxHistorySize = 32;
```

これはデータ構造とデータ数の制限の定義である。history は boolean[][] 型の 2 次元配列を格納するための Deque である。このデータ構造を扱うのは、セルの状態変更時、世代の更新時、巻き戻し時の 3 つの場合である。世代の更新時については前節で説明したため省略する。
セルの状態変更時について説明する。以下に抜粋した箇所を示す。

```
    public void changeCellState(int x, int y) {
        boolean[][] currentCells = new boolean[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                currentCells[i][j] = cells[i][j];
            }
        }
        history.push(currentCells);

        if (history.size() > maxHistorySize) {
            history.pollLast();
        }

        if (cells[x][y] == false) {
            cells[x][y] = true;
        } else {
            cells[x][y] = false;
        }
```

```
        fireUpdate();
    }
```

ここで用いたローカル変数の説明を図で行う。

変数	型	説明
x	int	操作対象のセルの横方向のインデックス
y	int	操作対象のセルの縦方向のインデックス
currentCells	boolean[][]	現在の盤面の状態のコピーを保持する二次元配列
i	int	ループ変数 (行方向)
j	int	ループ変数 (列方向)

まず、現在の盤面を保存するために必要な一時的に保持する配列 currentCells を定義し、セル 1 つずつコピーする。コピーできたら、ヒストリーに push する。このとき、ヒストリーに格納されている盤面の数が設定した 32 個よりも多ければ、ヒストリーの最後の要素を消去する pollLast を実行する。その後、指定された座標の生死状態を切り替えて最後に更新通知を行う。
次に、巻き戻りメソッドについて説明する。

```
    public void undo() {
        if (!history.isEmpty()) {
            boolean[][] previousCells = history.pop();
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    cells[i][j] = previousCells[i][j];
                }
            }
            fireUpdate();
        }
    }
```

盤面状態の巻き戻しを行う際は、previousCells に最新の記録をポップしてそれを一つずつコピーする。その後変更通知をする。

3.3 盤面の描画において、セルの境界線の位置を計算する方法・計算式

以下は、BoardView クラスの盤面の描画の機能を行う箇所を抜粋したものである。

```
    int w,h,l,m;

w = this.getWidth();
h = this.getHeight();

if(w>h) {
```

```
l=h;
}else {
l=w;
}
m = l/14;

for(int a = m; a < 14*m; a += m) {
  g.drawLine(m, a, 13*m, a);
  g.drawLine(a, m, a, 13*m);
}
```

ここで用いたローカル変数の説明を図で行う。

変数名	型	説明
w	int	ウィンドウの横の長さ
h	int	ウィンドウの縦の長さ
l	int	ウィンドウの縦横の短い方の長さを格納する
m	int	l を 14 で割った値 (セルの 1 辺の長さ)
a	int	ループのためのカウンタ変数

表 1: メソッド内で用いた変数の説明

まず、ウィンドウの縦と横の長さを受け取り、短い方を l に格納する。それを 14 で割ったものをセル 1 辺の長さとして扱う。ここで、14 で割っているのは、盤面が 12 段分あり、その上下左右に余白をもたせるためである。その後 for ループを使用して、横および縦のグリッド線を描画する。ウィンドウの端から、セル 1 個分から 13 個分目まで長さ l の感覚でそれぞれ縦横描画することで、1 段分余白をもった盤面の描画を行うことができる。

3.4 マウスカーソルの座標から対応するセルの座標を計算する方法・計算式

以下は、BoardModel クラスの、与えられた座標からセルの座標を求めるメソッドを抜粋したものである。

```
public void searchPoint(int x, int y, int w, int h) {
   int l, m, n = 12, o = 12;

   if (w > h) {
       l = h;
   } else {
       l = w;
   }
   m = l / 14;
   for (int a = 0; a < 12; a++) {
```

```
            if (a * m + m < x && x < a * m + 2 * m) {
                n = a;
            }
        }
        for (int a = 0; a < 12; a++) {
            if (a * m + m < y && y < a * m + 2 * m) {
                o = a;
            }
        }
        if (n == 12 || o == 12) {
            cell_x = 12;
            cell_y = 12;
        } else {
            cell_x = n;
            cell_y = o;
        }
    }
```

ここで用いた変数の説明を図で行う。なお、w,h,l,m は盤面の描画のメソッドと同様であるため省略する。

変数名	型	説明
cell_x	int	今クリックされたセルの x 座標
cell_y	int	今クリックされたセルの y 座標
n	int	一時的にセルの x 座標を表す変数。初期値は 12。
o	int	一時的にセルの y 座標を表す変数。初期値は 12。
a	int	m を用いたループ制御のカウンタ。

m を求めるところまでは先ほどと同様である。与えられた座標がそれぞれウィンドウの端から m 何個目分の範囲に入るのかをループを用いて探す。a が 0 から 11 の範囲で見つかればそれを n,o に一時的に代入する。なければ、初期値として設定した 12 のままである。n,o のどちらかが 12 のとき、その座標は盤面の外を指している。盤面の中を指している場合は cell_x と cell_y に n,o の値を入れる。

3.5 同セル内でドラッグした際に状態変更が行われないための処理

以下は、マウスドラッグ時に行う処理の記述を抜粋したものである。

```
   public void mouseDragged(MouseEvent e) {
int x = e.getX();
int y = e.getY();
boardmodel.searchPoint(x, y, getWidth(), getHeight());
if(boardmodel.getCellX()!=12 && boardmodel.getCellY()!=12) {
```

```
    if (boardmodel.getCellX() != cell_x_pre || boardmodel.getCellY() != cell_y_pre) {
         boardmodel.changeCellState(boardmodel.getCellX(), boardmodel.getCellY());
         cell_x_pre = boardmodel.getCellX();
         cell_y_pre = boardmodel.getCellY();
         repaint();
       }
 }
}
```

以下に用いた変数を図で説明する。

変数	型	説明
x	int	マウスドラッグが発生した際の X 座標
y	int	マウスドラッグが発生した際の Y 座標
cell_x_pre	int	1 つ前のセルの X 座標
cell_y_pre	int	1 つ前のセルの Y 座標

x と y の値を受け取ったら、まず、先ほどのセルを特定して cell_x と cell_y にそれぞれセルの座標を格納するメソッド SearchPoint を実行し、それらがどちらも 12 でない場合、すなわちドラッグされたのが盤面外でない場合、cell_pre と比較する。そして一致しないときに、生死状態の切り替えを行うメソッド ChangeCellState を呼び出す。その後、cell_pre を更新し、盤面を再描画する。なお、ここのメソッドで呼び出している getCell メソッドは、BoardModel クラスの cell_x と cell_y を得るためのものである。

3.6 新しいウィンドウを開く方法

新しいウィンドウが開かれるのは”New”ボタンを押したときである。

```
    newButton = new JButton("New");
        newButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                SwingUtilities.invokeLater(new Main());
            }
        });
        buttonPanel.add(newButton);
```

まず、newButton.addActionListener メソッドによって、New ボタンがクリックされたときに実行されるリスナー（ActionListener）が設定される。actionPerformed メソッドでは、Main クラスの新しいインスタンスを生成する処理が実行される。そして、run メソッドが呼び出され、新しいウィンドウを作成し表示するためのコードが実行される。以上の仕組みである。

4　Java プログラミングで学習したこと

今回学習した Java の特徴について、おそらく最も人気があるであろう python、1 改正の時に学習した C 言語と比較しながら説明する。

4.1　静的型付けと動的型付け

C 言語は静的型付け言語であり、変数の型は宣言時に指定されなければならない。これにより、コンパイル時に型エラーが検出される。Java も静的型付け言語であり、変数の型は事前に宣言される。型安全性が高く、コンパイル時に型の整合性が検査され、実行時の型エラーが少ない。Python は動的型付け言語であり、変数の型は実行時に決定される。型の宣言が不要で柔軟性があるが、実行時に型エラーが発生する可能性がある。

4.2　メモリ管理

C 言語ではプログラマがメモリ管理を明示的に行う必要がある。malloc や free などの関数を使用して動的メモリの確保と解放を手動で行う。Java はガベージコレクションを採用しており、プログラマがメモリを明示的に解放する必要はない。不要なオブジェクトは自動的に収集され、メモリリークのリスクが低減される。Python もガベージコレクションを使用し、メモリ管理は自動で行われる。プログラマは通常、メモリの明示的な解放を気にする必要はない。

4.3　オブジェクト指向プログラミング

C 言語は手続き型言語であり、オブジェクト指向プログラミングを直接サポートしていない。構造体や関数ポインタを使用してオブジェクト指向的なコーディングは可能だが、直感的ではない。Java はオブジェクト指向プログラミングを採用しており、クラスやオブジェクトの概念が中心。再利用性が向上し、メンテナンスが容易になる。Python もオブジェクト指向プログラミングをサポートし、クラスや継承、ポリモーフィズムなどの機能が豊富。シンプルな構文で効果的なオブジェクト指向プログラミングが可能。

4.4　マルチスレッドサポート

C 言語は標準ライブラリでマルチスレッドをサポートしているが、スレッドの管理や同期に関する操作はプログラマが直接行う必要がある。Java は言語仕様にスレッドサポートが組み込まれており、マルチスレッドプログラミングが比較的容易。スレッド同期やスレッドプールなどの機能が提供されている。Python もスレッドサポートを提供しているが、CPython では GIL（Global Interpreter Lock）があるため、同時に 1 つのスレッドしか実行されない。制約がある場合はマルチプロセスを検討する。

4.5　コードの移植性

C 言語はハードウェアやオペレーティングシステムに依存せず、移植性が高い。ただし、プラットフォームによってはコンパイラやライブラリの違いが影響を与えることがある。Java は「Write Once, Run Anywhere」の哲学に基づき、プラットフォームに依存しないバイトコードを生成。Java 仮想マシン（JVM）がインス

トールされている環境ならどこでも実行可能。Python も基本的には移植性があり、C 拡張モジュールやプラットフォーム依存のライブラリを使用する場合は注意が必要。バージョンの違いにも留意が必要。

4.6 まとめ

Java は静的型付け、オブジェクト指向、ガベージコレクションなどの特徴により、大規模で保守性と拡張性が求められるアプリケーション開発などに向いている。C 言語は低レベルな制御や高いパフォーマンスが求められる場面で使われ、Python はシンプルで読みやすい構文と動的型付けにより、迅速な開発と柔軟性が求められる場面で利用されることがある。それぞれの言語は異なる使用ケースに適しており、プロジェクトのニーズに合わせて選択することが重要である。

4.7 参考文献

Java 入門第 3 版 (中山清喬. 国本大悟)
サイト「Java ってどんな言語？その仕様から C 言語との違いまで徹底解説！」
URL:https://www.geekly.co.jp/column/cat-technology/1906_015/
サイト「Java と Python の違い」
URL:https://kinsta.com/jp/blog/python-vs-java/