# **J2Kad10D「可変長配列（ArrayListクラス）」 （実践編P.100「コレクション」、P.93「ArrayListクラスとジェネリクス」）**

Monsterクラスが準備されている。

1. ArrayListにMonsterを5匹格納して、格納したデータを表示する処理を通常のfor文を使って作成せよ。
2. ①で作成したfor文を拡張for文に変更せよ。

**ArrayList<Monster>の仕様（今回、使用する可能性のあるメソッド）**

|  |
| --- |
| **Monster** |
| name |
| Monster()  toString() |

|  |  |
| --- | --- |
| メソッド | 仕様 |
| boolean add(Monster data) | 配列にデータを追加する。 |
| int size() | 配列のデータ数を返す。 |
| Monster get(int index) | 配列のindex番目のデータを返す。 |

**課題完成時の画面**

データを格納します！

add：マルマイン

add：イワンコ

add：モクロー

add：レアコイル

add：トランセル

データを表示します！

get：マルマイン

get：イワンコ

get：モクロー

get：レアコイル

get：トランセル

Monsterを追加するごとに名前を表示する。

System.out.println("add：" + モンスターの名前);

Monsterの名前を先頭から順に表示する。

System.out.println("get：" + モンスターの名前);

# **J2Kad10C「ラッパークラス」 （実践編P.96「ラッパークラスを用いた基本型の格納」、P.98「オートボクシングとオートアンボクシング」）**

J2Kad10Dと同等の処理をint型に対して行え。なおint型（基本型）はArrayListに格納できないので注意。

1. ArrayListにint型変数（乱数で0～99）を5つ格納して表示する処理を通常のfor文を作成せよ。
2. ①で作成したfor文を拡張for文に変更せよ。

**Integerクラスの仕様（今回、使用する可能性のあるメソッド）**

|  |  |
| --- | --- |
| メソッド | 仕様 |
| static Integer valueOf(int i) | 整数iの値を持つIntegerクラスを返す。 |
| int intValue() | Integerクラスの持つ整数値を返す。 |

**課題完成時の画面**

データを格納します！

add：63

add：71

add：40

add：11

add：16

データを表示します！

get：63

get：71

get：40

get：11

get：16

int型変数を追加するごとに値を表示する。

System.out.println("add：" + 値);

格納された値を先頭から順に表示する。

System.out.println("get：" + 値);

# **J2Kad10B「連想配列（HashMapクラス）」（実践編P.104「マップコレクション」）**

HashMapを使ってECCバーガーのメニューを表示する処理を作成せよ。必要であれば拡張for文を使うこと。

**HashMap<String, Integer>の仕様（今回、使用する可能性のあるメソッド）**

|  |  |
| --- | --- |
| メソッド | 仕様 |
| value put(String key, Integer value) | ハッシュマップにデータ（key, value）を追加する。 |
| Set<String> keySet() | ハッシュマップのkeyセット（文字列配列のようなもの）を返す。 |
| Integer get(String key) | ハッシュマップのkeyに対応する値（value）を返す。 |

**ECCバーガーのメニュー 課題完成時の画面**

|  |  |
| --- | --- |
| 品名 | 値段（円） |
| ハンバーガー | 150 |
| チーズバーガー | 180 |
| ビッグマック | 410 |

ECCバーガーへようこそ！メニューを表示します！

ビッグマック：410円

チーズバーガー：180円

ハンバーガー：150円

# **J2Kad10A「双方向リスト（LinkedList）」（実践編P.104「LinkedList」）**

**リスト1**はLinkedListにMonsterを格納・削除する処理である。**課題完成時の画面**を参考にデータの表示と各コマンドに対応する処理を作成せよ。

**コマンドごとの処理**

|  |  |
| --- | --- |
| コマンド | 処理 |
| 0：addFirst | リストの先頭にMonsterを追加し、「先頭に～（名前）を追加した！」と表示する。 |
| 1：addLast | リストの最後にMonsterを追加し、「最後に～（名前）を追加した！」と表示する。 |
| 2：removeFirst | リストの先頭のMonsterを削除し、「先頭の～（名前）を削除した！」と表示する。 |
| 3：removeLast | リストの最後のMonsterを削除し、「最後の～（名前）を削除した！」と表示する。 |

**LinkedList<Monster>の仕様（今回、使用する可能性のあるメソッド）**

|  |  |
| --- | --- |
| メソッド | 仕様 |
| void addFirst(Monster data) | リストの先頭にデータを追加する。 |
| void addLast(Monster data) | リストの最後にデータを追加する。 |
| Monster getFirst() | リストの先頭のデータを返す。 |
| Monster getLast() | リストの最後のデータを返す。 |
| Monster removeFirst() | リストの先頭のデータを削除する（戻り値は削除したデータ）。 |
| Monster removeLast() | リストの最後のデータを削除する（戻り値は削除したデータ）。 |
| boolean isEmpty() | リストにデータがないときtrueを返す。 |
| int size() | リストのデータ数を返す。 |
| Monster get(int index) | リストのindex番目のデータを返す。 |

**リスト1：LinkedListを使った処理**

**import java.util.LinkedList;** // インポート（追加すること）

import java.util.Scanner;

public class J2Kad11S {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

**LinkedList<Monster> list = new LinkedList<>();**  // LinkedListの宣言（追加すること）

while(true) {

// データの表示

**データの表示を作成すること**

// コマンド入力

System.out.print("どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞");

int cmd = in.nextInt();

if (cmd < 0) break;

**各コマンドに対応する処理を作成すること**

System.out.println();

}

}

}

**課題完成時の画面**

現在のリスト：

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**0**

先頭にピジョンを追加した！

現在のリスト：ピジョン ->

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**0**

先頭にスピアーを追加した！

現在のリスト：スピアー -> ピジョン ->

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**1**

最後にトランセルを追加した！

現在のリスト：スピアー -> ピジョン -> トランセル ->

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**2**

先頭のスピアーを削除した！

現在のリスト：ピジョン -> トランセル ->

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**3**

最後のトランセルを削除した！

現在のリスト：ピジョン ->

どうしますか？（0：addFirst、1：addLast、2：removeFirst、3：removeLast、-1：終了）＞**-1**

# **J2Kad10S「ArrayListを作ろう！（MyArrayクラス）」※J2Kad10Dをコピーして改造 （実践編P.103「ArrayList」）**

ArrayListと同等の処理を行うMyArrayクラスを作成し、J2Kad10Dと同等の処理を作成せよ。なお、拡張for文は使えないので通常のfor文で記述すること。

ArrayList<Monster> list = new ArrayList<>(); → MyArray list = new MyArray(); に変更する

|  |
| --- |
| **MyArray** |
| - array : Monster[] |
| + MyArray()  + add(data : Monster) : void  + get(i : int) : Monster  + size() : int |

**課題完成時の画面**

（J2Kad10Dと同じ）

# **J2Kad10X「LinkedListを作ろう！（MyListクラス）」※J2Kad10Aをコピーして改造 （実践編P.104「LinkedList」）**

LinkedListと同等の処理を行うMyListクラスを作成し、J2Kad10Aと同等の処理を作成せよ。なお、拡張for文は使えないので通常のfor文で記述すること。

LinkedList<Monster> list = new LinkedList<>(); → MyList list = new MyList(); に変更する

**リスト1：MyNodeクラス（先に作成すること）**

public class MyNode {

public Monster data; // Monsterへの参照

public MyNode prev; // 前のノードへの参照

public MyNode next; // 次のノードへの参照

public MyNode(Monster data, MyNode prev, MyNode next) {

this.data = data;

this.prev = prev;

this.next = next;

}

}

**リスト2：MyListクラス（双方向リスト）**

|  |
| --- |
| **MyList** |
| - dummy : MyNode |
| + MyList()  + size() : int  + isEmpty() : boolean  + get(i : index) : Monster  + addFirst(data : Monster) : void  + addLast(data : Monster) : void  + getFirst() : Monster  + getLast() : Monster  + removeFirst() : Monster  + removeLast() : Monster |

public class MyList {

private MyNode dummy; // ダミーノード

**必要なメソッドは各自で考えて定義すること**

}

**課題完成時の画面**

（J2Kad10Aと同じ）

# **単方向リスト　←検索**

リストとは複数のノードのつながりで構成されるデータ構造のことです。ノードは、データと次のノードへの参照（next）で構成され、nextが次のノードを示すことによって、ノードをつなげていきます。動的にデータのつながりを作るのに適しています。

|  |  |
| --- | --- |
| データ | next |

|  |  |
| --- | --- |
| データ | next |

|  |  |
| --- | --- |
| データ | next |

|  |  |
| --- | --- |
| データ | next |

# **双方向リスト（J2Kad10A、J2Kad10X）　←検索**

双方向リストは前のノードへの参照（prev）も持っているリストのことです。

MyNode

MyList

dummy

|  |  |  |
| --- | --- | --- |
| prev | ダミー | next |

|  |  |  |
| --- | --- | --- |
| prev | データ | next |

|  |  |  |
| --- | --- | --- |
| prev | データ | next |

|  |  |  |
| --- | --- | --- |
| prev | データ | next |

リスト本体（データ部分）

←先頭 最後→

リストの先頭（ダミーの次）にデータを追加するには以下のようにします（ダミーの次にノードAがあり、ダミーとノードAの間にノードBを挿入する）

1. ノードBを生成する（ノードBのprevにダミー、nextにノードAを指定）。
2. ノードAのprevをノードBにする。
3. ダミーのnextをノードAにする。