● J2Kad11D「可変長配列（ArrayList クラス）」

Monster クラスが準備されている。

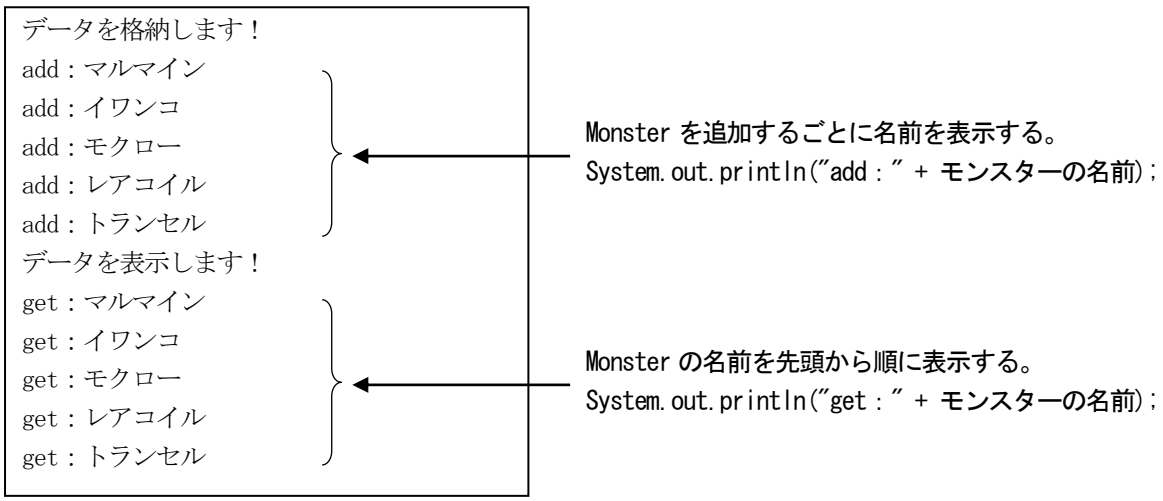
① ArrayList に Monster を 5 匹格納して、格納したデータを表示する処理を作成せよ。ただし拡張 for 文は使わないこと。
② ①で作成した for 文を拡張 for 文に変更せよ。

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

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。

Monster
name
Monster() toString()

課題完成時の画面



● J2Kad11C「ラッパークラス」

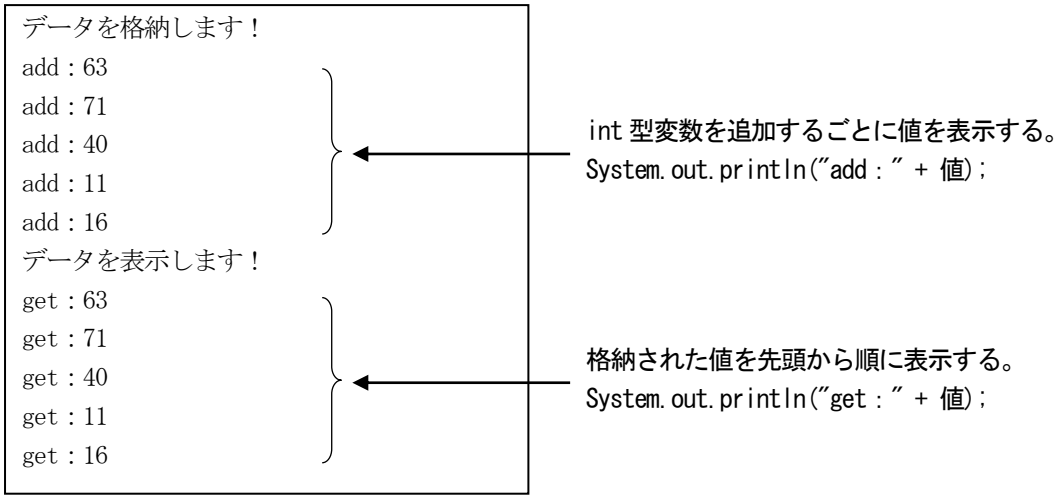
J2Kad11D と同等の処理を int 型に対して行え。

① ArrayList に int 型変数（乱数で 0～99）を 5 つ格納して格納したデータを表示する処理を作成せよ。ただし拡張 for 文は使わないこと。
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



● J2Kad11B「連想配列（HashMap クラス）」

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。ECC バーガーのメニューは以下の通り。なお、必要であれば拡張 for 文を使うこと。

ECC バーガーのメニュー

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

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。

課題完成時の画面

```
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
```

● J2Kad11A「ArrayList を作ろう（MyArray クラス）」※J2Kad11D①の main メソッドをコピー

リスト１をもとにArrayList と同等の処理を行う MyArray クラスを作成せよ。J2Kad11D①（通常の for 文）の段階のコードをコピーし動作確認せよ。

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

リスト１：MyArray クラス（作成すること）

```
public class MyArray {
    private Monster[] array;
    public MyArray() {
        ⋮
    }
    public void add(Monster data) {
        ⋮
    }
    public Monster get(int i) {
        ⋮
    }
    public int size() {
        ⋮
    }
}
```

課題完成時の画面

（J2Kad11D と同じ）

● J2Kad11S「双方向リスト（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
```

● J2Kad10X「LinkedList を作ろう！（MyList クラス）」※J2Kad11S の main メソッドをコピー

LinkedList と同等の処理を行う MyList クラスを作成せよ。MyList は双方向リスト（←検索）で作成すること。
J2Kad11S の main メソッドをコピーして作成するが、拡張 for 文を使っている箇所は通常の for 文に変更すること。

```
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）

```
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
}
```

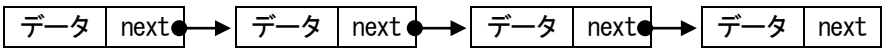
MyList
- dummy : MyNode
+ MyList()
+ size() : int
+ isEmpty() : boolean
+ get(i : index) : Monster
+ addFirst(data : Monster) : void
+ addLast(data : Monster) : void
+ getFirst() : Monster
+ getLast() : Monster
+ removeFirst() : Monster
+ removeLast() : Monster

課題完成時の画面

（J2Kad11S と同じ）

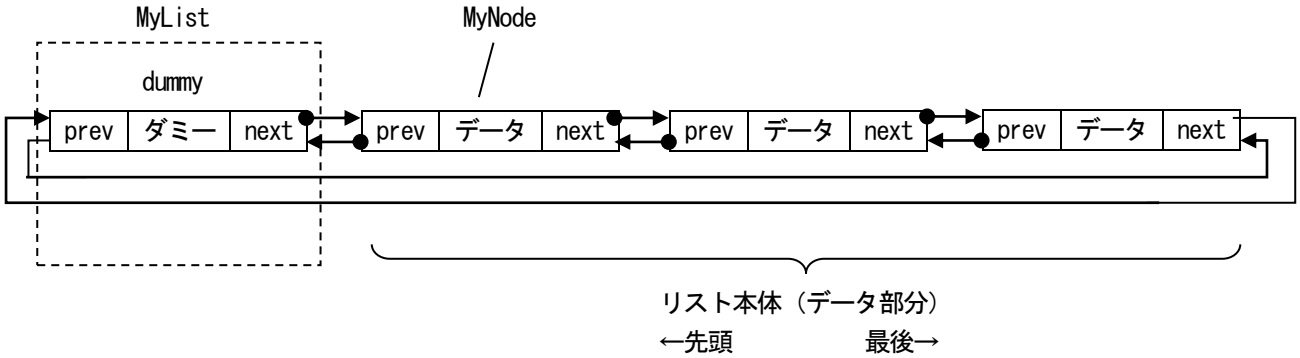
● 単方向リスト　←検索

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。



● 双方向リスト（J2Kad11X で作成）　←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。



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。