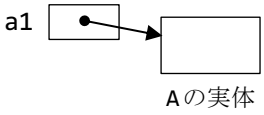
アルゴリズムとデータ構造 2020　第９回 演習課題　「連結リスト」

1. Java においてクラスは参照型であり，クラス型の変数やフィールドは，その実体ではなくメモリ内でのアドレスを保持する。例えば，次のようにクラス A，B，C を定義した場合を想定する。

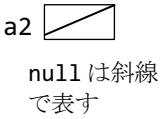
```
class A { }    class B { A a; }    class C { C c; }
```

このとき，次のようなコードを考えるとそれぞれその下のように図示できる。

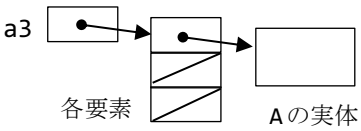
```
A a1 = new A();
```



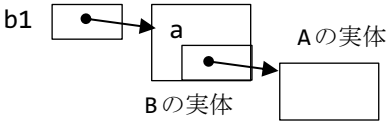

```
A a2 = null;
```




```
A[] a3 = new A[3];
a3[0] = new A();
```



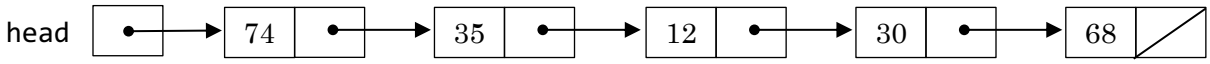

```
B b1 = new B();
b1.a = new A();
```



同様にして，以下のコードを図示せよ。

```
C c1 = new C();
C c2 = new C();
c1.c = c2;
c2.c = null;
```

2. **連結リスト**は動的なデータ構造の一種であり，下図に示すようにデータを格納する**ノード（またはセル）**を**リンク**と呼ばれるつながりで連結する。これによって，データ数の制限をなくし，挿入や削除を容易にする。下記のプログラムは連結リストの構造を理解するためのものである。下図の連結リストを構築する処理を main メソッドに記入せよ。先頭から全ノードをたどって表示する処理 printAll も完成させよ。




```
/* Node.java */
public class Node {
   public int data;   // データ
   public Node next;  // 連結（つながり）

   public Node(int data) {
      this.data = data;
      next = null;
   }
}

/* List.java */
public class List {
   public Node head;  // 先頭ノード

   public List() {
      head = null;
   }

   public void printAll() {
      System.out.print("[ ");

      // リストの中のノードを 1 つずつたどる
      for (Node n = head;
            n != null; n = n.next) {


      }
      System.out.println("]");
   }
}
```

```
/* Program.java */
public class Program {
   public static void main(String[] args) {

      // ノードを連結して連結リストを構築せよ
      // 直接データ構造を操作できるように
      // すべてのフィールドを公開にしてある

      List list = new List();
      Node n1 = new Node(74);
      Node n2 = new Node(35);
      Node n3 = new Node(12);
      Node n4 = new Node(30);
      Node n5 = new Node(68);

      list.head =









      list.printAll();

      // Node n = list.search(30);
      // if (n != null)
      //    System.out.println(n.data);
   }
}
```

3. 下記のプログラムは，第 8 回の 1.と同じ機能のスタックを連結リストで実現したものである。このアルゴリズムでは，リストの先頭（head）にデータを格納し，先頭から順に取り出す。適切に空欄を埋めてプログラムを完成させ，動作を確認せよ。

```
/* Node.java */
public class Node {  // リスト要素（ノード）

   // 同一パッケージからのみアクセス可能
   int data;  // 格納データ
   Node next; // 次のノードへのリンク

   Node(int data) {



   }
}
```

```
/* Stack.java */
public class Stack {

   // 連結リストの先頭ノードを指す
   private Node head;

   public Stack() {
      head = null;
   }

   // リストの先頭にノードを加える
   public void push(int data) {
      Node n = new Node(data);

      // n の後ろに今のリストをつなげる
                = head;

      head =
    }
```

```
   // リストの先頭からノードを取り出す
   public int pop() throws Exception {
      // データがない場合をチェック
      if (               ) {

         throw new Exception("empty!");
      }
      // data を取り出し，新しい head を設定
                  = head.data;

      head =

      return data;
   }

   public void printAll() {
      System.out.print("[ ");

      // リストの中のノードを 1 つずつたどる
      for (Node n = head;
           n != null; n = n.next) {



      }
      System.out.println("]");
   }
}
```

```
/* Program.java */
/* メインプログラムは第 8 回の 1.のものを
   そのまま利用する。ただし，new Stack(5)
   は new Stack() に書き換える。
*/
```

4. 3.のプログラムにおいて，push と pop のリンクのつなぎ替えの処理を（4 コマ漫画のような感じで）図示せよ。さらに，スタック内のデータ数 n に対する push と pop の計算量（オーダー）を考察せよ。

5. 2.のクラス List に先頭から順にデータを探索するメソッドを追加して動作を確認し，計算量を考察せよ（これも**線形探索**の一種である）。データが見つからなかった場合は，null を返すようにすること。

```
public Node search(int data) {




}
```

6. 【発展】連結リストでは，次のノードが存在しないことを表す値として null を用いることが一般的だが，下記のプログラムのように特別なノードを用いることもできる。これを用いると，探索においてループ内の比較処理の回数を減らすことができる。これは，探索範囲の最終要素にあらかじめ探索しているデータを入れておく**番兵**（sentinel）と呼ばれる手法の一種である。

```
/* Node.java */
public class Node {
   public int data;   // データ
   public Node next;  // 連結（つながり）

   public Node(int data) {
      this.data = data;
   }
}

/* List.java */
public class List {
   public Node head;  // 先頭ノード

   // 連結先がないことを示すノード
   final public Node END = new Node(-1);

   public List() {
      head = null;
   }

   public void printAll() {
      System.out.print("[ ");

      // リストの中のノードを 1 つずつたどる
      for (Node n = head;
            n != END; n = n.next) {


      }
      System.out.println("]");
   }

   public Node search(int data) {

      END.data = data;
      Node n = head;

      while (            ) { // 比較が 1 回

         n = n.next;
      }


   }
}
```

```
/* Program.java */
public class Program {
   public static void main(String[] args) {

      // ノードを連結して連結リストを構築せよ
      // 直接データ構造を操作できるように
      // すべてのフィールドを公開にしてある

      List list = new List();
      Node n1 = new Node(74);
      Node n2 = new Node(35);
      Node n3 = new Node(12);
      Node n4 = new Node(30);
      Node n5 = new Node(68);

      list.head =




      n5.next = list.END

      list.printAll();

      Node n = list.search(30);
      if (n != list.END)
         System.out.println(n.data);
   }
}
```