

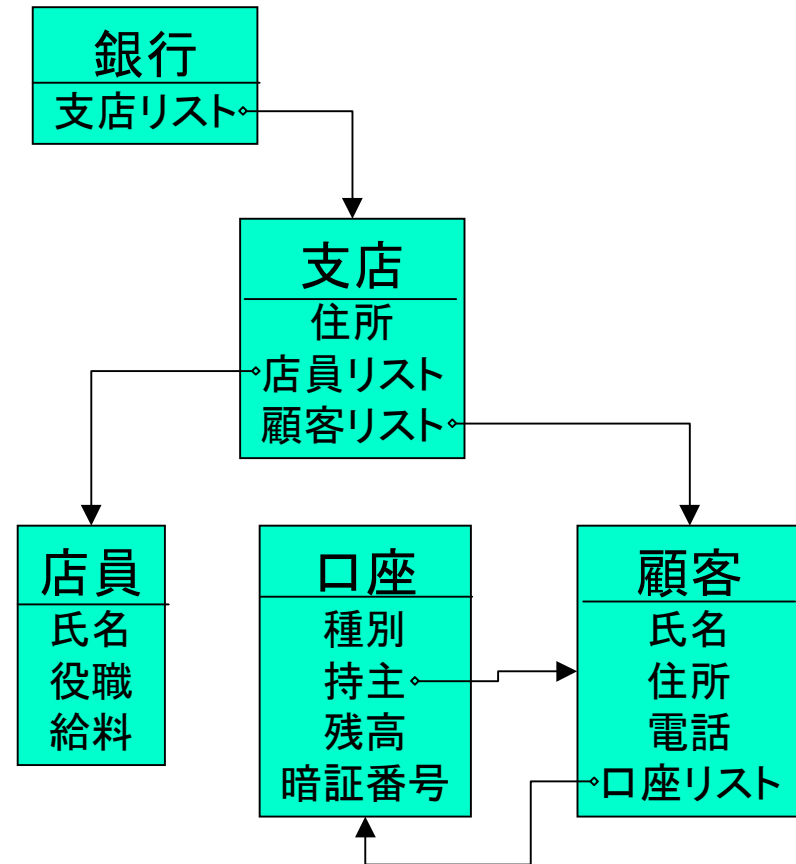
計算機プログラミングI

第11回 2003年1月9日(木)

- (第10回の補足)
- 再帰的データ構造
 - ◇ リスト構造
- 課題: 描画エディタの作成

データ構造

- データ構造:
データ型の設計
(例: 銀行口座)
 - ◇ データが持つ情報の種類
(例: 口座は持主・残高・・・)
 - ◇ データとデータの関係
(例: 支店は口座と店員を管理)
 - ◇ どのように使われるか
(例: 入金処理・口座開設・・・)



データ構造

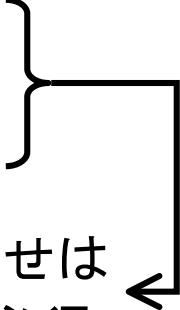
- Java言語のデータ型

- ◇基本型 (a.k.a. 原始型): 整数(int), 実数(double), ...

- ◇配列型 — 例: 整数の配列, Turtleの配列, ...

- ◇オブジェクト型 — 例: Turtle, Complex, ...

複数データの組み合わせは
実用的プログラムで必須



- オブジェクト指向言語におけるデータ構造

- ◇クラスの設計; 特にインスタンス変数の型

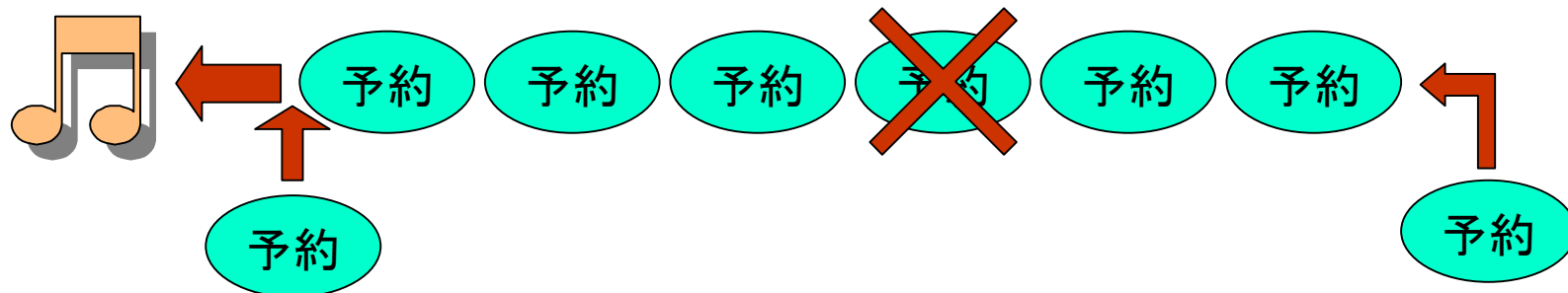
例題: リスト構造

• カラオケの予約リスト

◇ 予約が一行に並んだもの + 追加・削除機能

- 予約を追加する
- 先頭の予約を演奏
- 予約を取り消す
- 先頭に予約を割り込ませる

◇ cf. 配列: 個数が固定...追加・削除が難しい

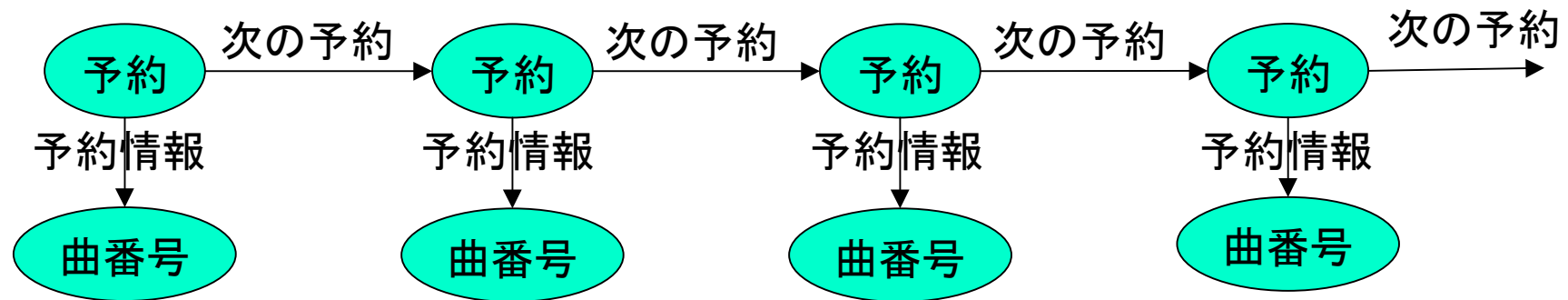


データ構造としてのリスト構造

・鎖状に並んだデータ



- ◇一つの輪が一つのデータを持つ
- ◇一つの輪は隣の輪だけを知っている
- ◇追加・挿入・削除: 輪をつなぐ・切る操作



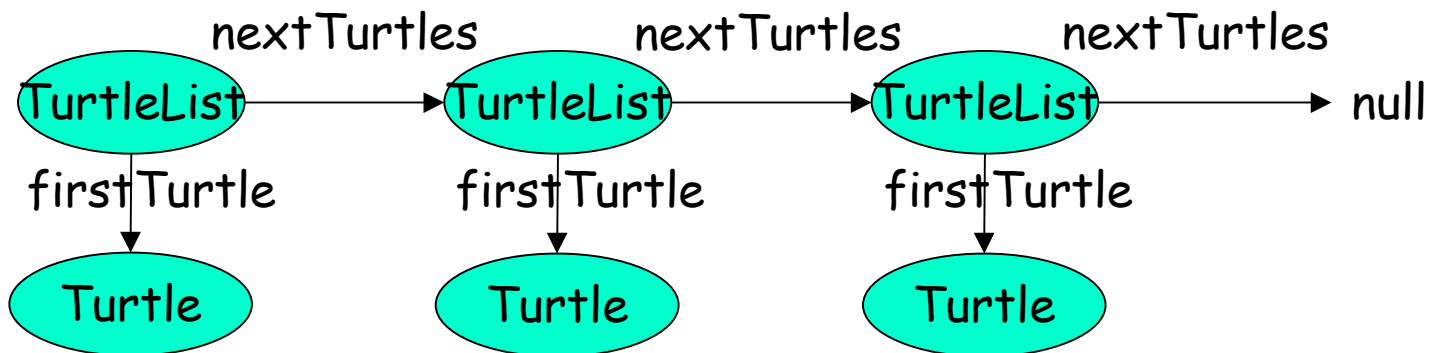
例題: タートルのリスト

- プログラム ListDemo
- 画面上にタートルが置かれる
- 機能: 画面上のボタンをクリックして
 - ◇ タートルをリストに追加
 - ◇ リスト上のタートルを前進
 - ◇ リスト上のタートルを回転
 - ◇ リストからタートルを削除
- (実演)

リスト構造: クラス定義

```
public class TurtleList {  
    public Turtle firstTurtle;  
    public TurtleList nextTurtles;  
    public TurtleList(Turtle f, TurtleList n) {  
        firstTurtle = f;  
        nextTurtles = n;  
    }  
}
```

先頭のタートル
続くリスト要素

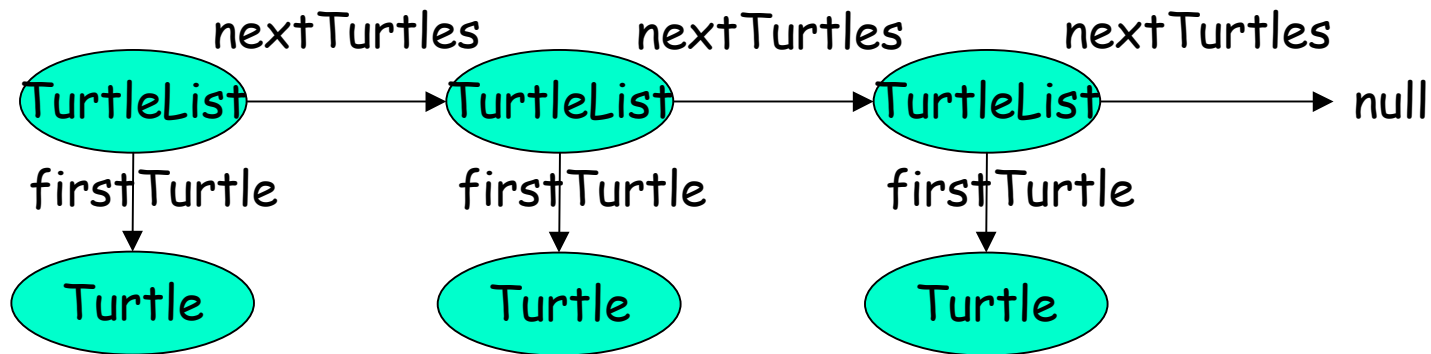


リスト構造: クラス定義

```
public class TurtleList {  
    public Turtle firstTurtle;  
    public TurtleList nextTurtles;  
    public TurtleList(Turtle f, TurtleList n) {  
        firstTurtle = f;  
        nextTurtles = n;  
    }  
}
```

先頭のタートル
続くリスト要素

自分自身の定義を使用
→再帰的なデータ構造



リスト構造の操作: リストを作る

```
public class ListDemo {  
    public static void main(String[] args) {  
        ウィンドウを作る(略)  
        TurtleList l = null;  
        while (true) {  
            if (「追加」ボタンが押された) {  
                int x = (略), y = (略);  
                Turtle m = new Turtle(x,y,0);  
                mを画面に表示する(略)  
                l = new TurtleList(m,l);  
            } else  
                他のボタンの処理(略)  
        }  
    }  
}
```

リスト構造の操作: リストを作る

```
public class ListDemo {  
    public static void main(String[] args) {  
        ウィンドウを作る(略)  
        TurtleList l = null;  
        while (true) {  
            if (「追加」ボタンが押された) {  
                int x = (略), y = (略);  
                Turtle m = new Turtle(x,y,0);  
                mを画面に表示する(略)  
                l = new TurtleList(m,l);  
            } else  
                他のボタンの処理(略)  
        }  
    }  
}
```

「何も参照していない」値
→ 空のリストを表わす
(null: 全ての参照型に
共通する特別な値)

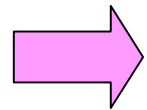
リスト構造の操作: リストを作る

```
public class ListDemo {  
    public static void main(String[] args) {  
        ウィンドウを作る(略)  
        TurtleList l = null;  
        while (true) {  
            if (「追加」ボタンが押された) {  
                int x = (略), y = (略);  
                Turtle m = new Turtle(x,y,0);  
                mを画面に表示する(略)  
                l = new TurtleList(m,l);  
            } else  
                他のボタンの処理(略)  
        }  
    }  
}
```

「何も参照していない」値
→ 空のリストを表わす
(null: 全ての参照型に
共通する特別な値)

すでにあるリストを
「続くリスト」とする
リスト要素を作る
→ 要素の追加

リスト構造の操作: リストを作る

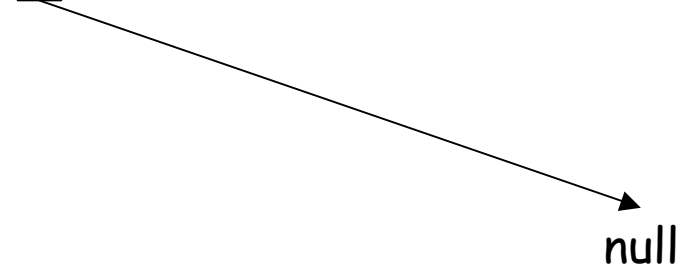


```
TurtleList l = null;
```

```
l = new TurtleList(m,l);
```

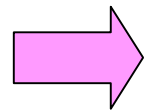
```
l = new TurtleList(m,l);
```

```
l = new TurtleList(m,l);
```



リスト構造の操作: リストを作る

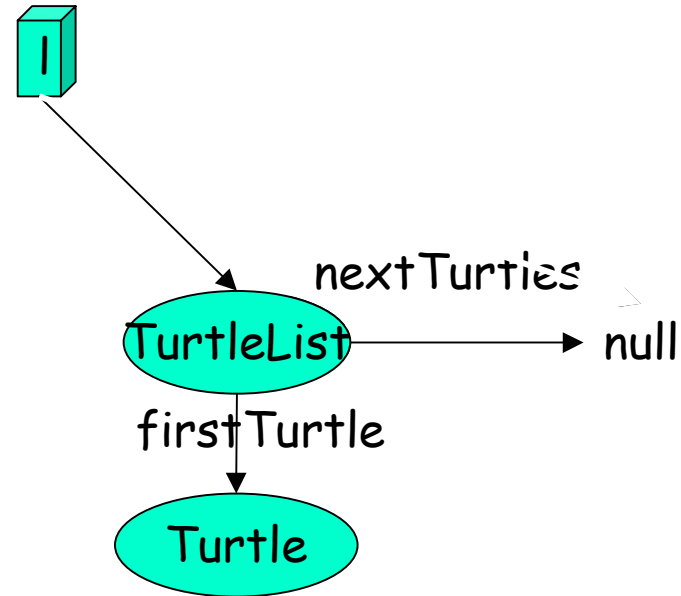
TurtleList l = null;



l = new TurtleList(m,l);

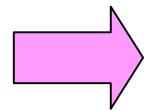
l = new TurtleList(m,l);

l = new TurtleList(m,l);



リスト構造の操作: リストを作る

```
TurtleList l = null;
```

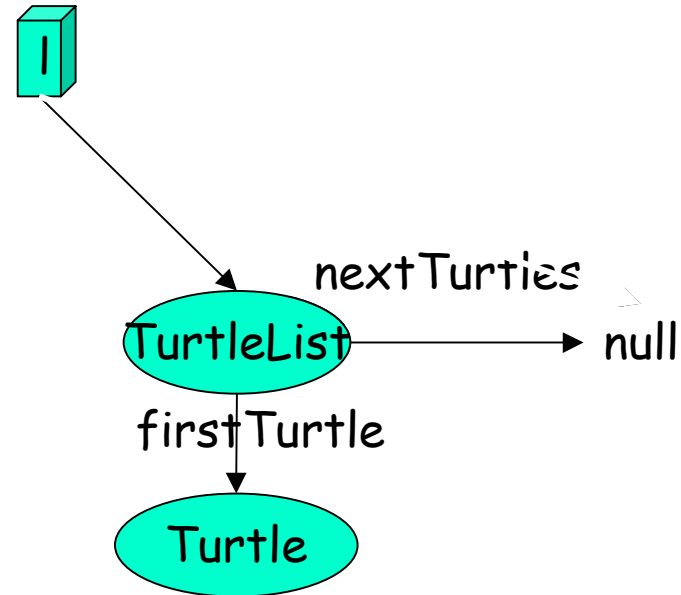


```
l = new TurtleList(m,l);
```

```
l = new TurtleList(m,l);
```

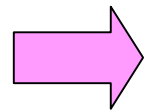
```
l = new TurtleList(m,l);
```

```
m = new Turtle(x,y,0);
```



リスト構造の操作: リストを作る

TurtleList l = null;

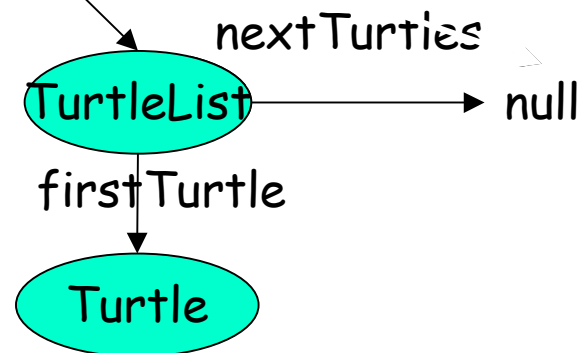
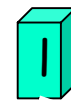


l = new TurtleList(m, l);

l = new TurtleList(m, l);

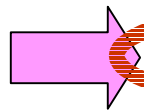
l = new TurtleList(m, l);

m = new Turtle(x, y, 0);



リスト構造の操作: リストを作る

```
TurtleList l = null;
```

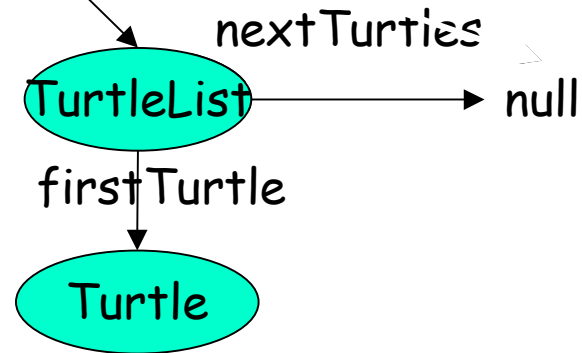


```
l = new TurtleList(m,l);
```

```
m = new Turtle(x,y,0);
```

```
l = new TurtleList(m,l);
```

```
l = new TurtleList(m,l);
```



リスト構造の操作: リストを作る

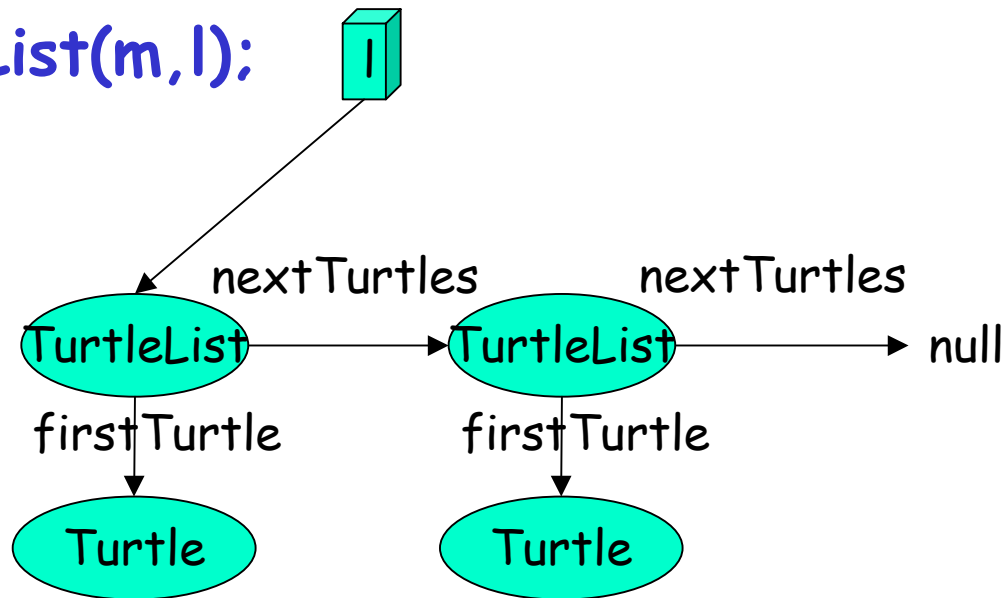
```
TurtleList l = null;
```

```
l = new TurtleList(m,l);
```

➡

```
l = new TurtleList(m,l);
```

```
l = new TurtleList(m,l);
```

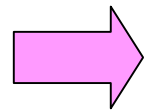


リスト構造の操作: リストを作る

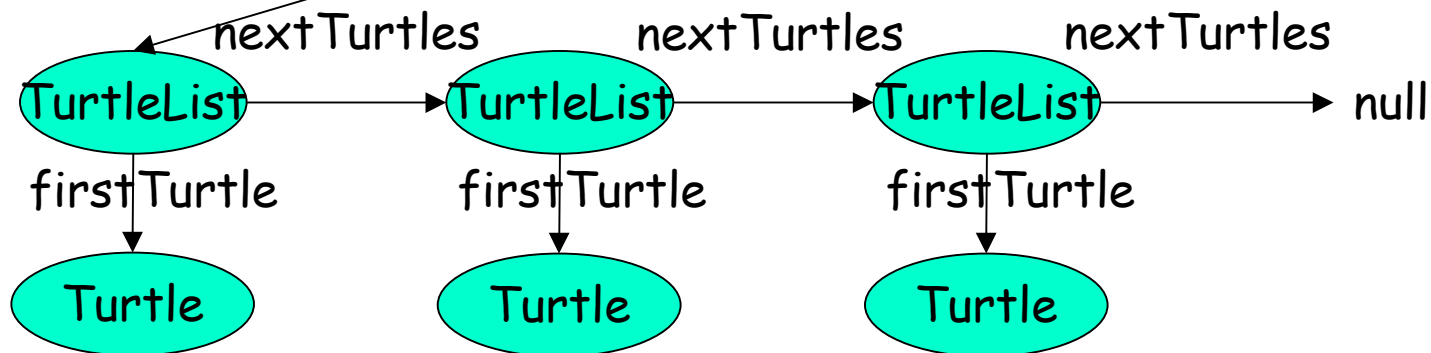
```
TurtleList l = null;
```

```
l = new TurtleList(m,l);
```

```
l = new TurtleList(m,l);
```



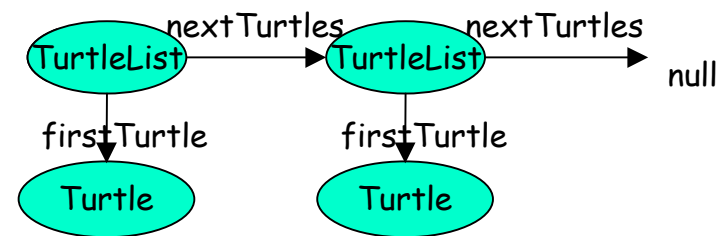
```
l = new TurtleList(m,l);
```



リスト構造の操作: 繰り返し

- 再帰的なメソッド呼び出し (※for文などでも可能)
- 例: リストの全ての要素に対して何かをする
= リストの先頭に何かをする
+ 続くリストの全ての要素に対して何かをする

```
public class TurtleList {
    インスタンス変数
    コンストラクタの定義
    public void forwardAll(int s) {
        firstTurtle fd(s);
        if (nextTurtles != null) {
            nextTurtles.forwardAll(s);
        }
    }
}
```

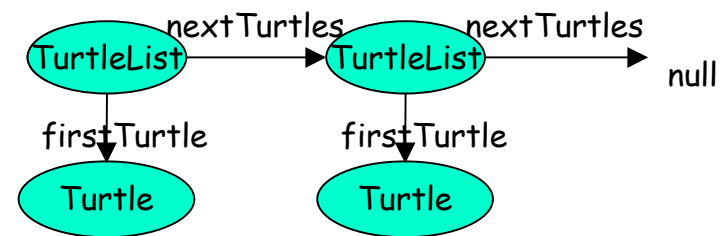


リスト構造の操作: 繰り返し

- 再帰的なメソッド呼び出し (※for文などでも可能)
- 例: リストの全ての要素に対して何かをする
= リストの先頭に何かをする
+ 続くリストの全ての要素に対して何かをする

```
public class TurtleList {
  インスタンス変数
  コンストラクタの定義
  public void forwardAll(int s) {
    firstTurtle fd(s);
    if (nextTurtles != null) {
      nextTurtles.forwardAll(s);
    }
  }
}
```

再帰的なメソッド呼出

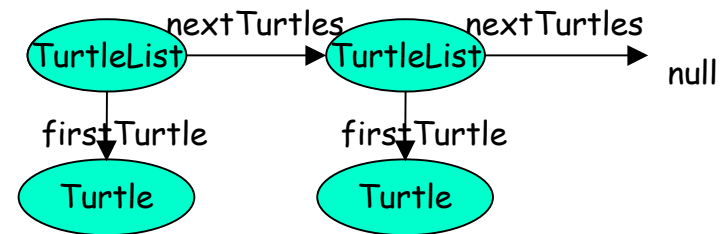


リスト構造の操作: 繰り返し

- 再帰的なメソッド呼び出し (※for文などでも可能)
- 例: リストの全ての要素に対して何かをする
= リストの先頭に何かをする
+ 続くリストの全ての要素に対して何かをする

```
public class TurtleList {  
    インスタンス変数  
    コンストラクタの定義  
    public void forwardAll(int s) {  
        firstTurtle fd(s);  
        if (nextTurtles != null) {  
            nextTurtles.forwardAll(s);  
        }  
    }  
}
```

リストの最後でない場合は
繰り返す



練習: リスト構造

11-1 (タートルの回転): 「前進」とほとんど同じ

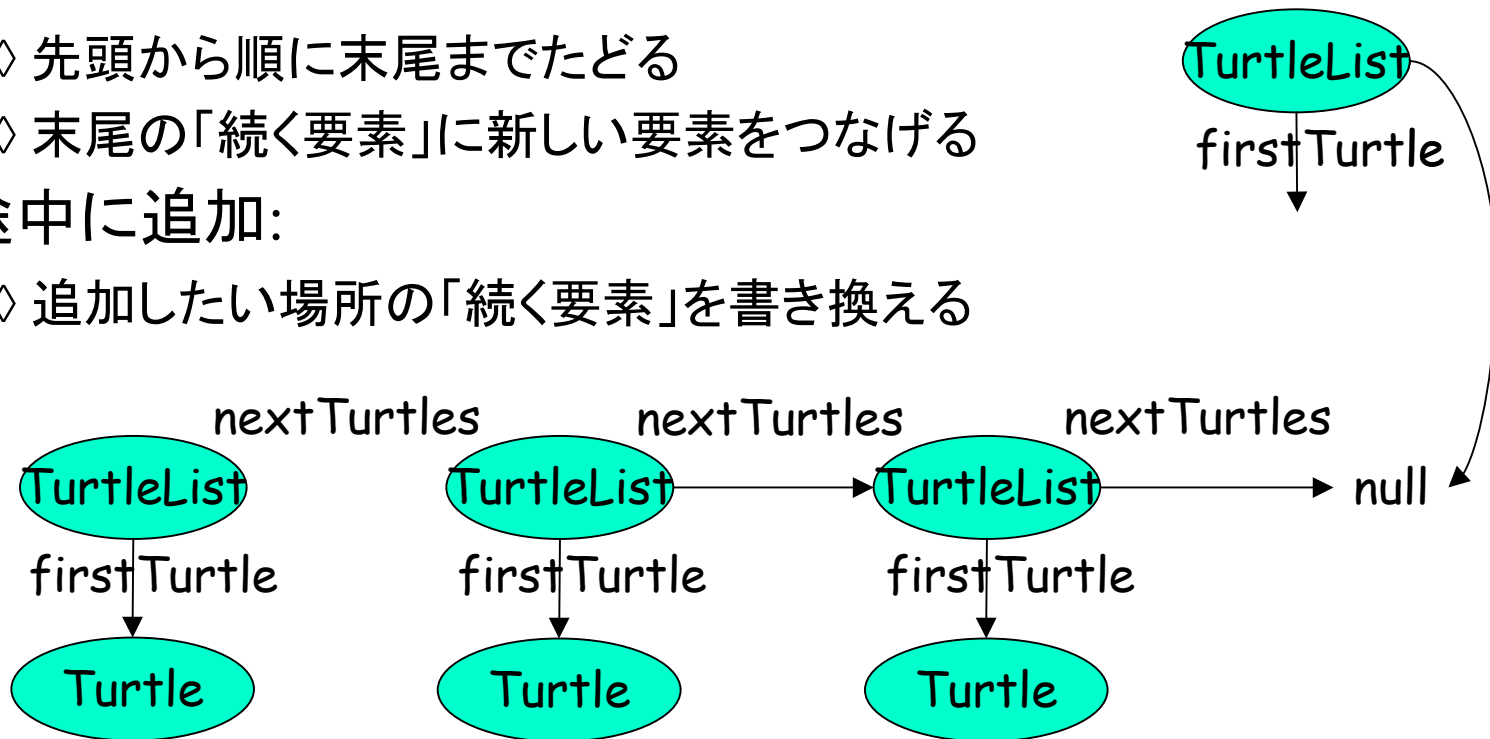
11-2 (リストの長さ)*: 「前進」や「回転」とほぼ同じ
値を返すことに注意

11-3 (最上位のタートル):

(参考プログラム: TurtleList.java, ListDemo.java,
ClickableTurtleFrame.java)

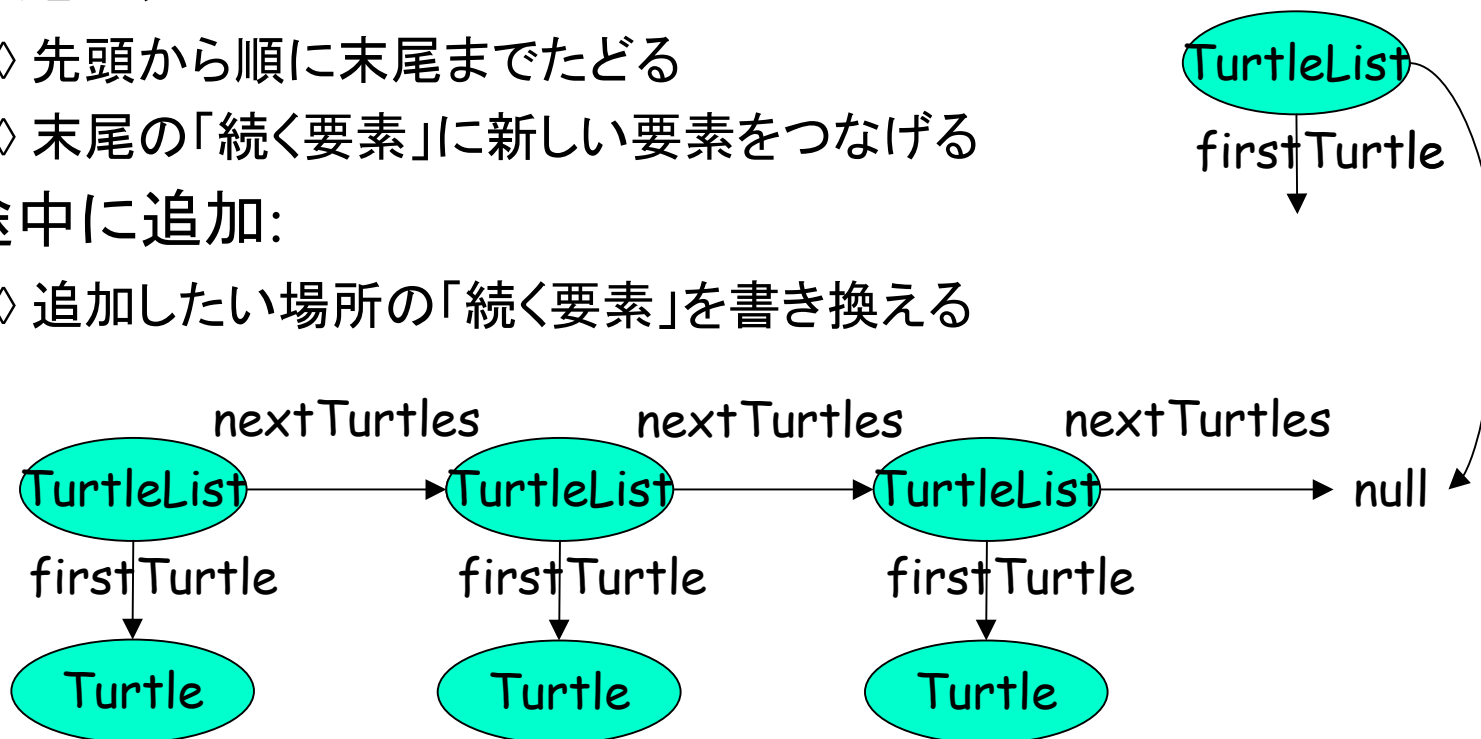
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



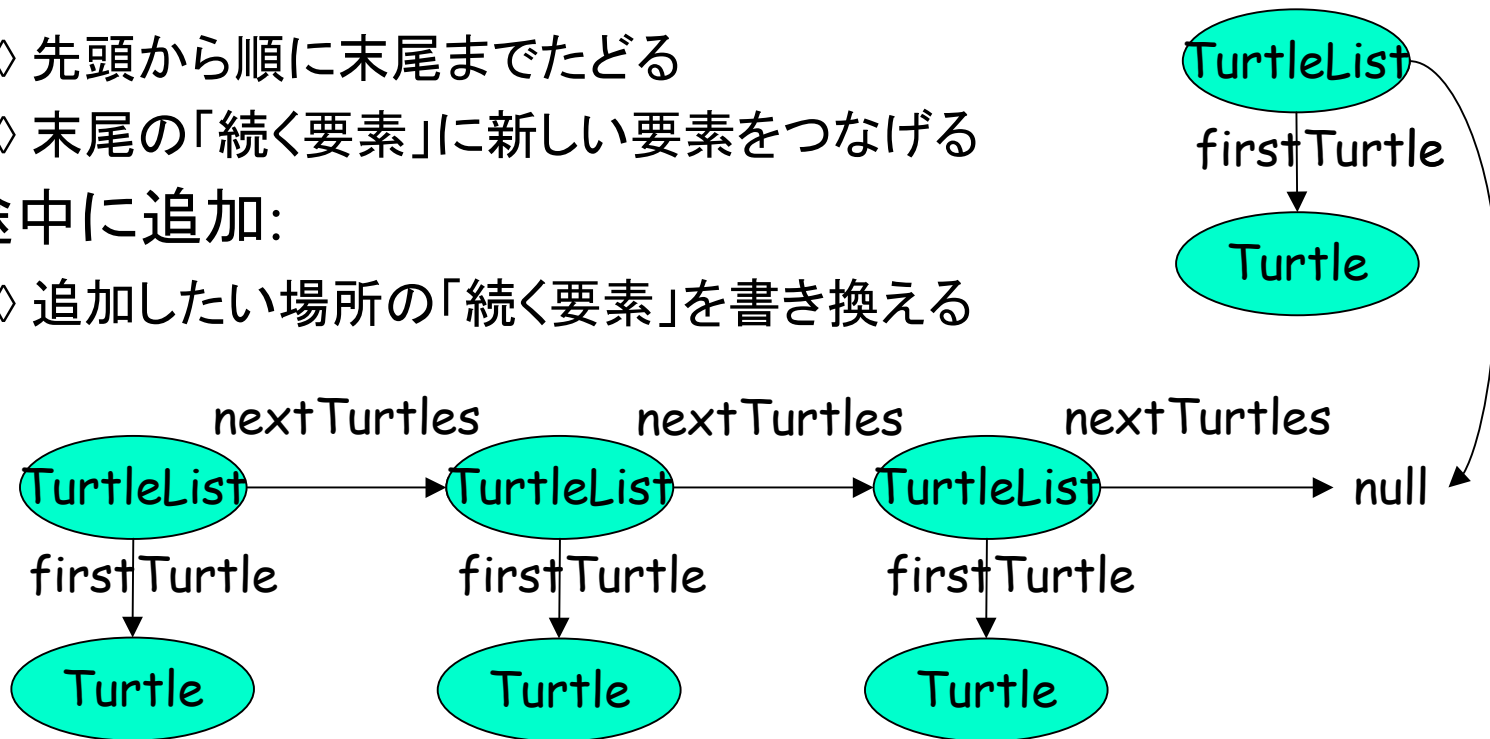
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



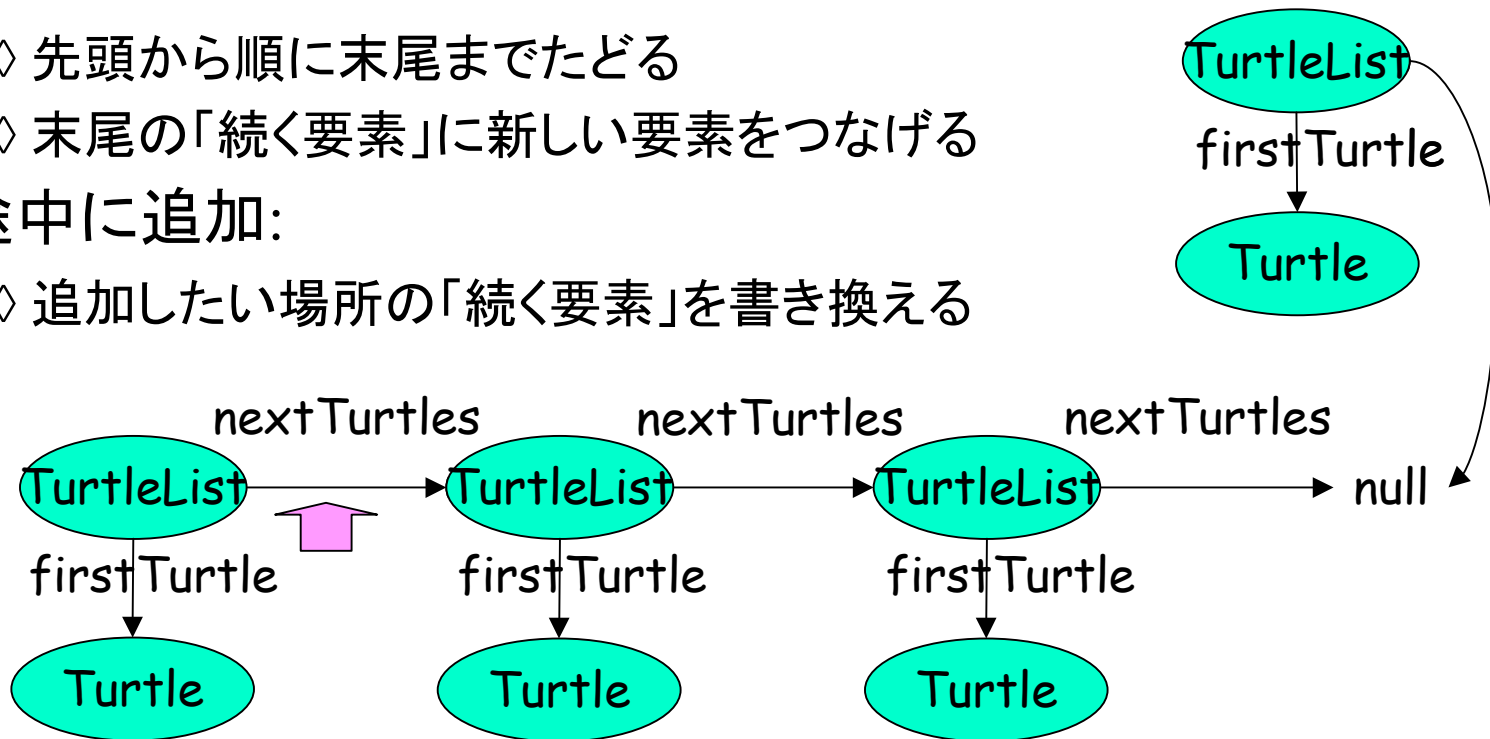
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



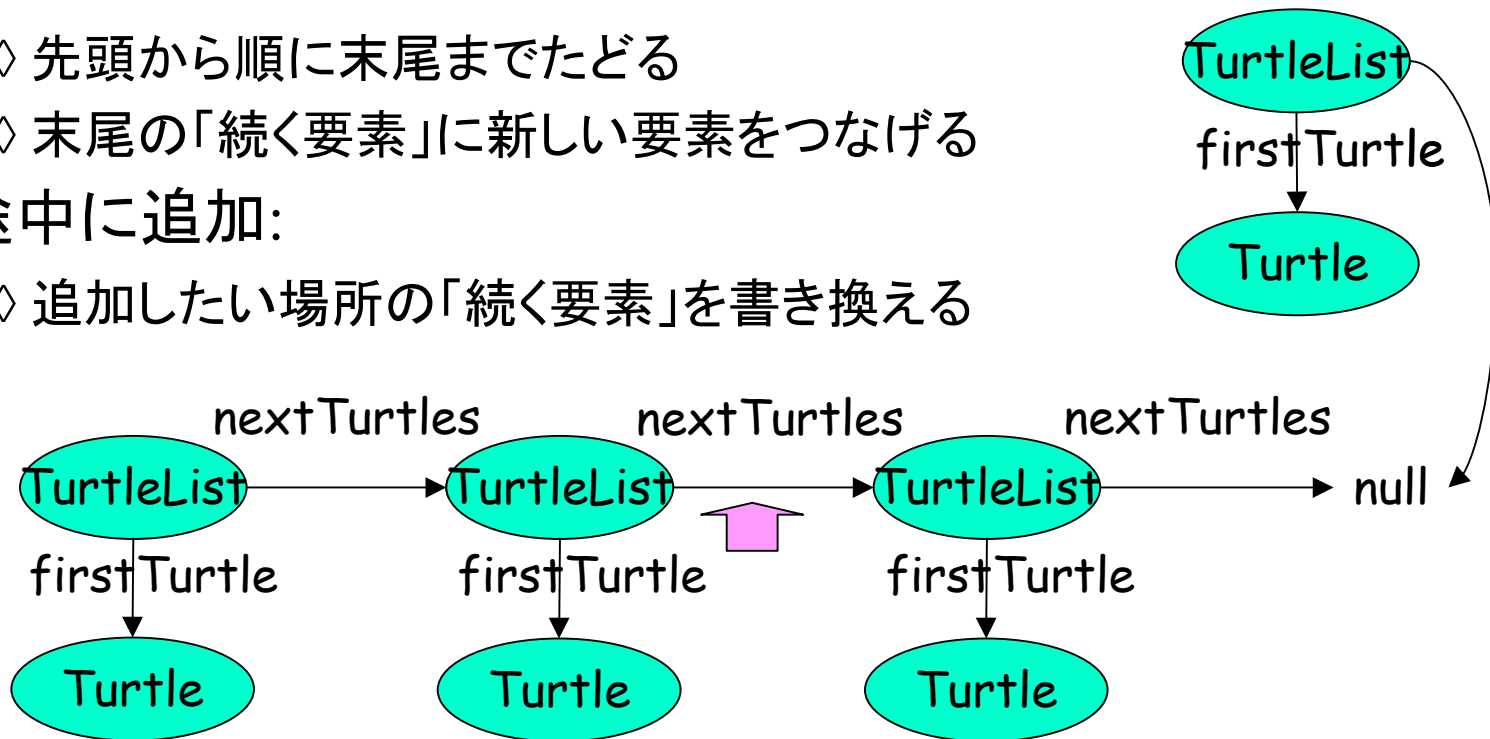
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



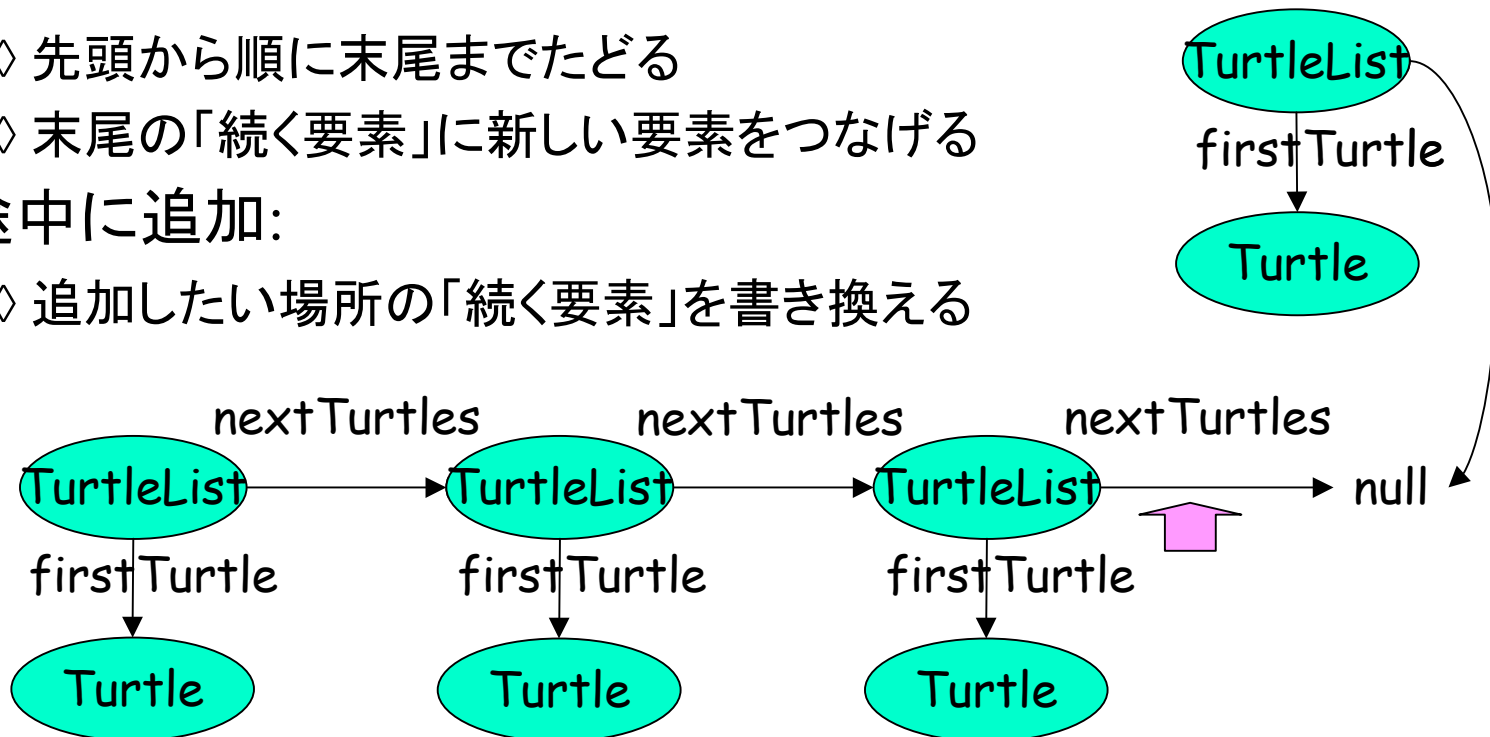
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



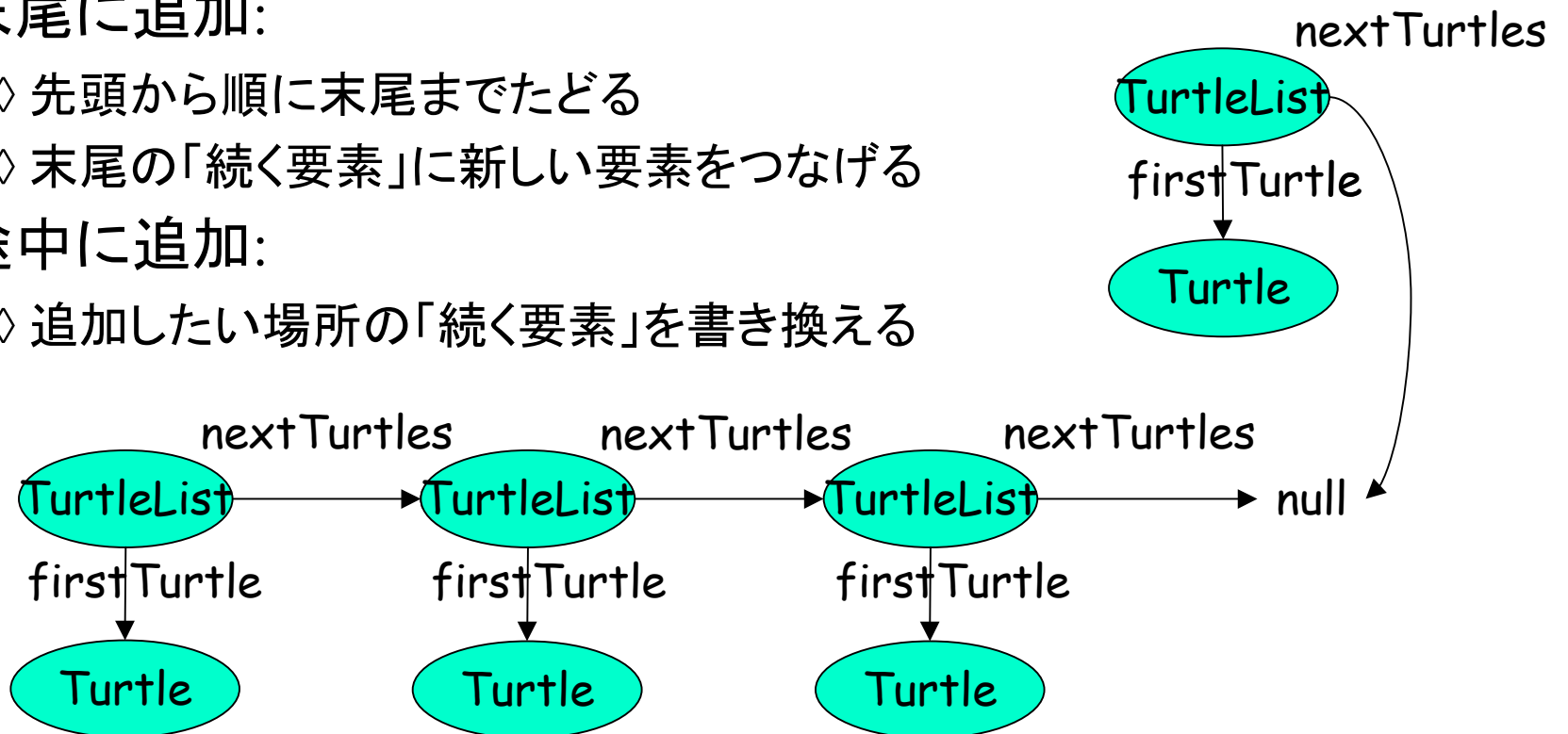
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



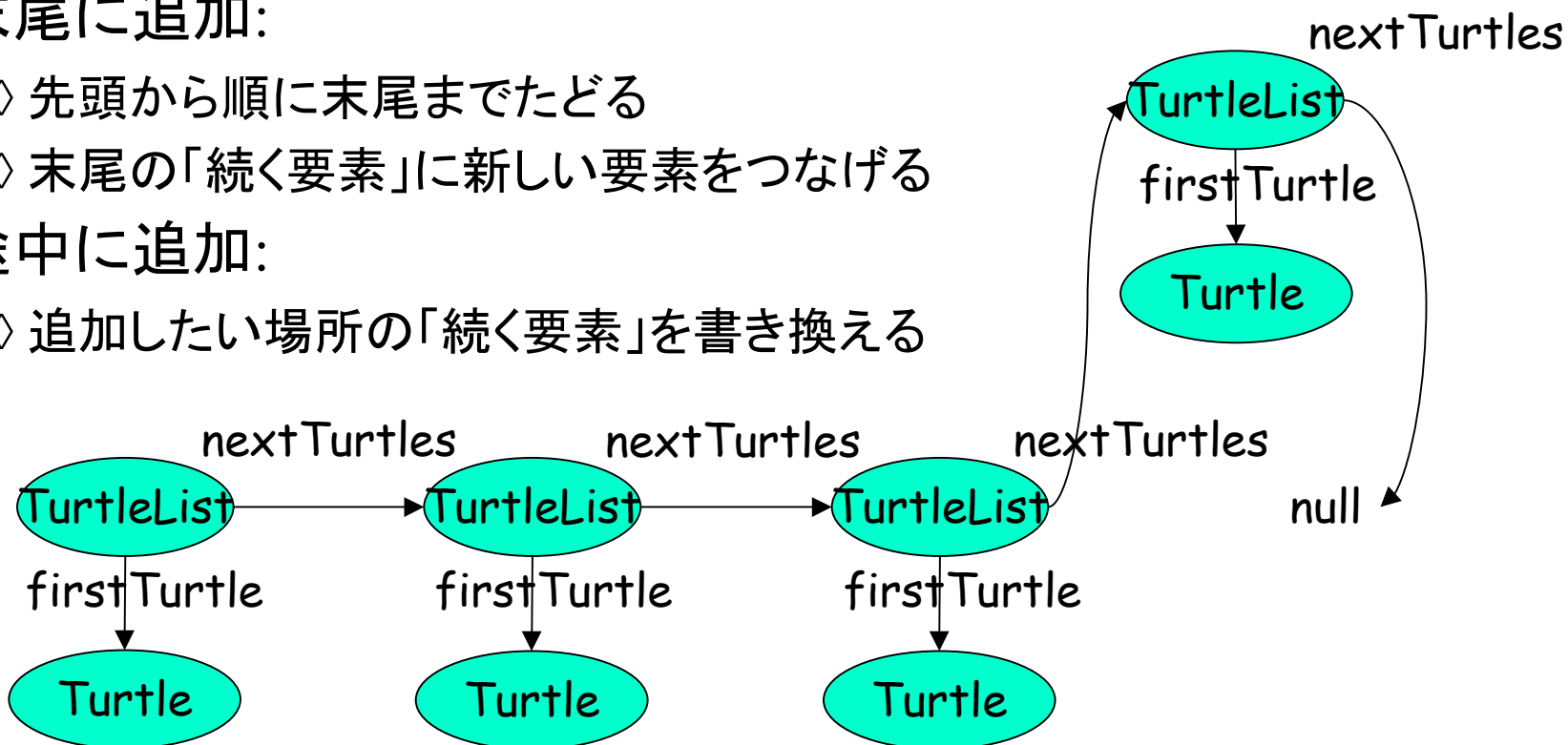
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



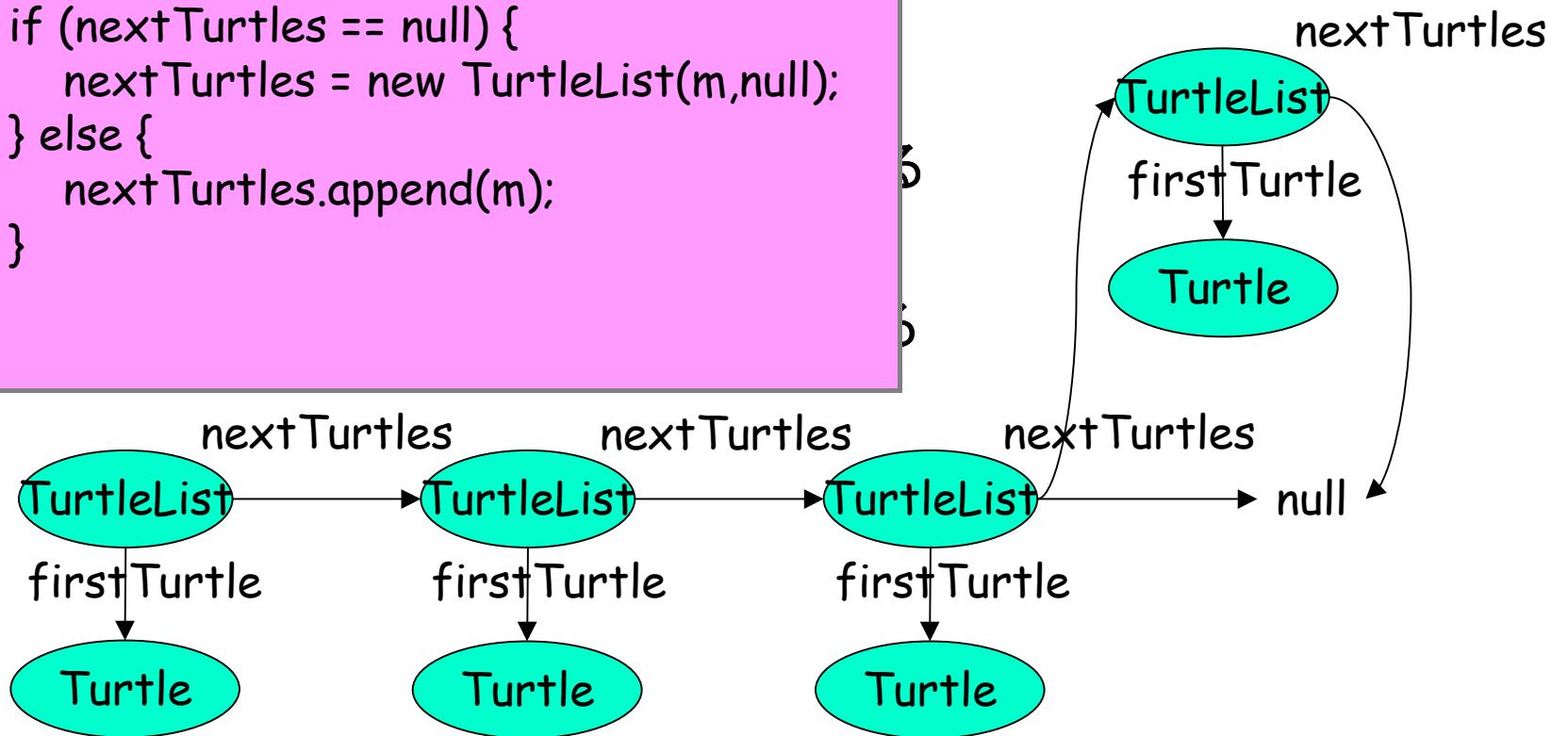
リスト構造の操作: 要素の追加

- 先頭に追加: 新しい要素を作るだけ
- 末尾に追加:
 - ◇ 先頭から順に末尾までたどる
 - ◇ 末尾の「続く要素」に新しい要素をつなげる
- 途中に追加:
 - ◇ 追加したい場所の「続く要素」を書き換える



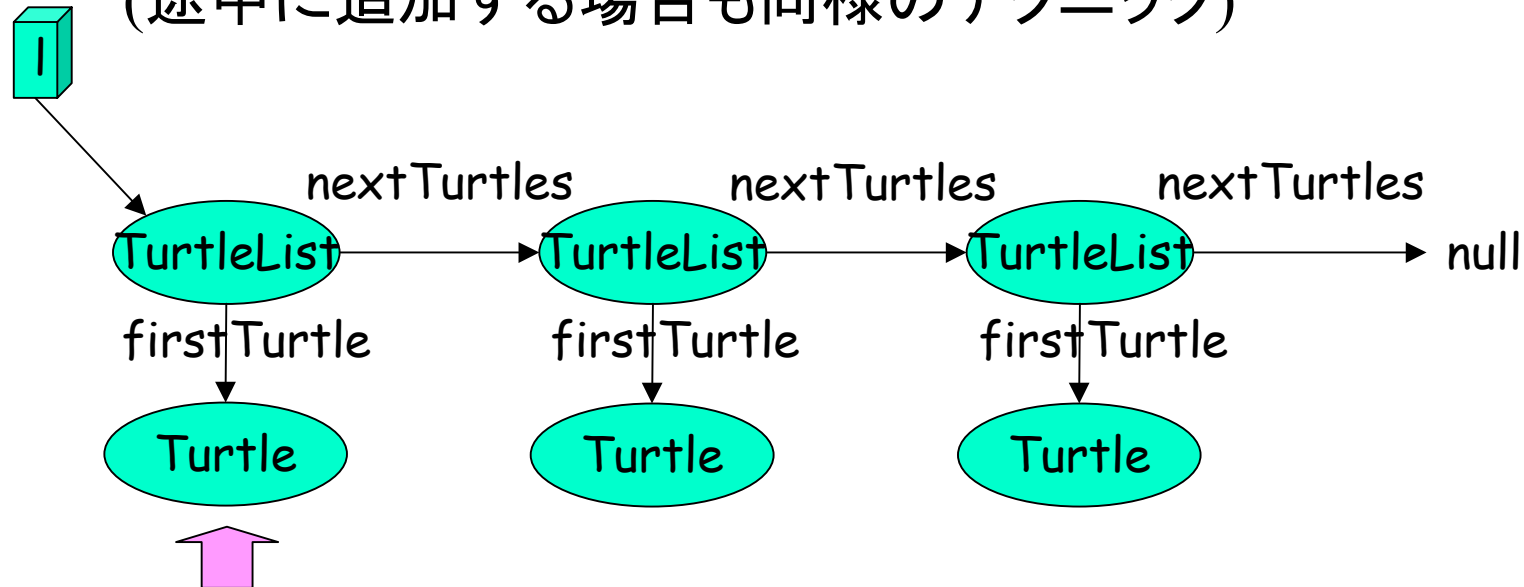
リスト構造の操作: 要素の追加

```
public class TurtleList {
    インスタンス変数・コンストラクタ・他のメソッド(略)
    public void append(Turtle m) {
        if (nextTurtles == null) {
            nextTurtles = new TurtleList(m,null);
        } else {
            nextTurtles.append(m);
        }
    }
}
```



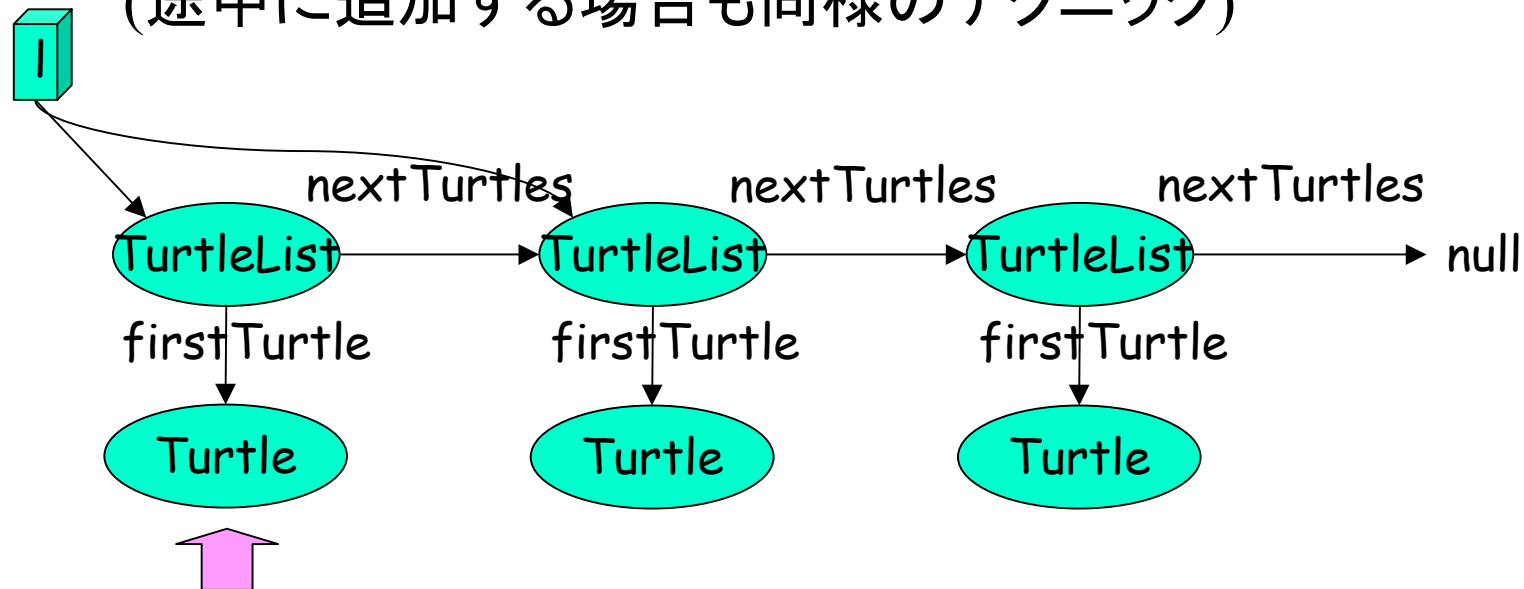
リスト構造の操作: 要素を削除する

- 先頭から削除: 「続くリスト要素」を先頭にする
- 途中を削除:
 - ◇ 削除する要素の手前の「続くリスト要素」を書き換える
 - ◇ プログラム上のテクニック:
(途中に追加する場合も同様のテクニック)



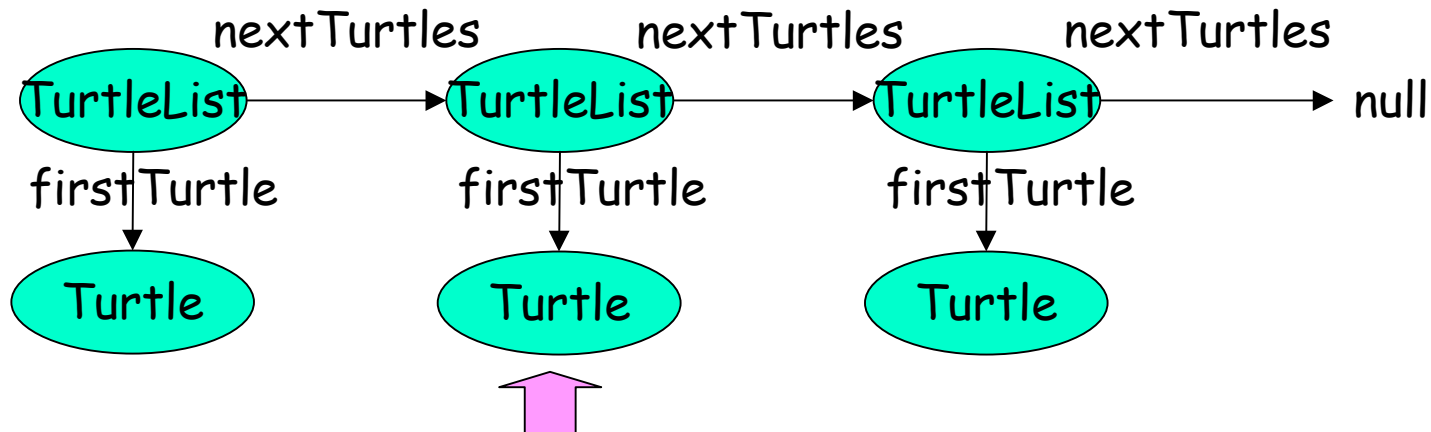
リスト構造の操作: 要素を削除する

- 先頭から削除: 「続くリスト要素」を先頭にする
- 途中を削除:
 - ◇ 削除する要素の手前の「続くリスト要素」を書き換える
 - ◇ プログラム上のテクニック:
(途中に追加する場合も同様のテクニック)



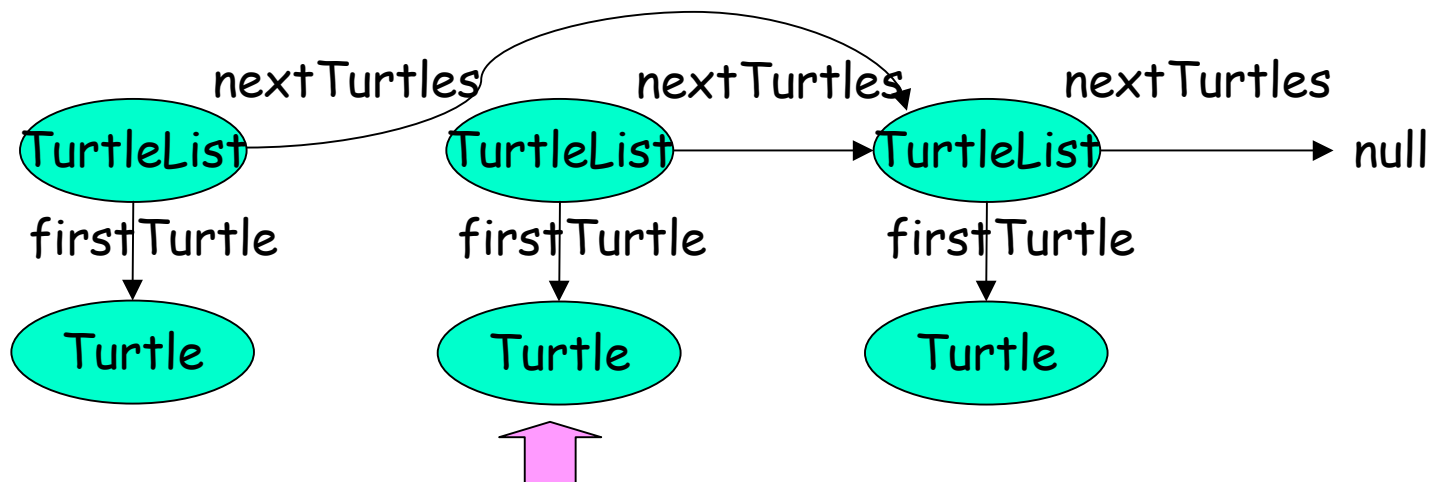
リスト構造の操作: 要素を削除する

- 先頭から削除: 「続くリスト要素」を先頭にする
- 途中を削除:
 - ◇ 削除する要素の手前の「続くリスト要素」を書き換える
 - ◇ プログラム上のテクニック:
(途中に追加する場合も同様のテクニック)



リスト構造の操作: 要素を削除する

- 先頭から削除: 「続くリスト要素」を先頭にする
- 途中を削除:
 - ◇ 削除する要素の手前の「続くリスト要素」を書き換える
 - ◇ プログラム上のテクニック:
(途中に追加する場合も同様のテクニック)



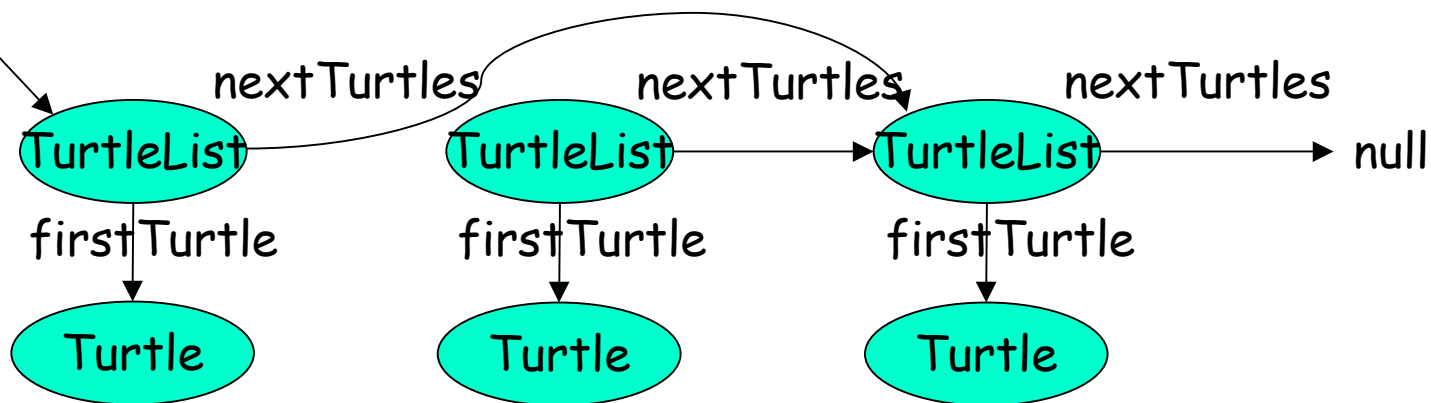
リスト構造の操作: 要素を削除する

```
public TurtleList delete(Turtle m) {  
    if (firstTurtle == m) {  
        return nextTurtles;  
    } else {  
        nextTurtles = nextTurtles.delete(m);  
        return this;  
    }  
}
```

自身が削除される要素だったら
次の要素を返す

`l = l.delete(m);`

ニック)



練習: リストへの挿入と削除

11-4 (最上のタートルの削除):

リストの途中から削除する例

11-5 (特定の場所への追加):

deleteメソッドと同様。

(cf. deleteは削除されたリストを返すメソッド)

11-6 (配列との比較)

第2回課題: 描画エディタ

提出スクリプト(コマンド cp1report2)についてはwebページの説明を参照せよ。

- 期限: 1月22日(水)
- 提出物と提出方法: a~eの各項目について
 - ◇ コンパイル可能なプログラムファイルをスクリプト実行によって提出、および
 - ◇ 課題の考察および解いた方法の説明を以下のどちらかの方法で提出
 - HTML形式で作ったファイルをスクリプト実行によって提出、あるいは
 - 紙に書かれたものをレポートボックスへ提出
- 注意事項:
 - ◇ **考察・説明を紙で提出する場合でも、プログラムはスクリプト実行によって提出すること。**
 - ◇ プログラムは1つのクラスを1つのファイルとし、項目(a~e)それぞれについて、どのファイルが対応しているかを示した索引ファイルと一緒に提出せよ。考察・説明をHTMLファイルで提出する場合は、その先頭に索引を書け。
 - ◇ 常識の範囲を越えた類似部分のあるレポートがあった場合は、追加の面接試験を行う場合や、当該レポートの評価を0点にすることがある。
 - ◇ 全ての項目を提出しなくてもよい。
 - ◇ 複数の項目をまとめた場合には、どの項目をどのようにまとめているかが分かるように明記すること。明記されていない場合は、全て提出されていないと見做される場合がある。
 - ◇ 提出物の中心は考察および解いた方法の説明である。(プログラムは説明が正しいことの証明に過ぎないので、それだけを提出してはならない。)
 - ◇ レポートの読みやすさ・独自性も採点の対象である。