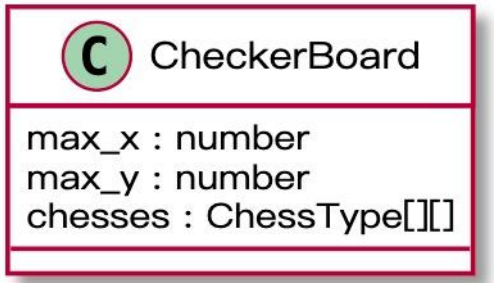
迭代器模式(Iterator)

场景1：棋盘示例

- 假设一个棋类游戏，黑白两色的棋子，在棋盘内部，棋子是用一个二维数组来存储。
- 有很多计算需要对棋盘中的棋子进行遍历：




```
for (int y = 0; y < checkerBoard.getMax_y(); y++)
    for (int x = 0; x < checkerBoard.getMax_x(); x++)
        System.out.print(checkerBoard.getChess(x, y) + " ");
```

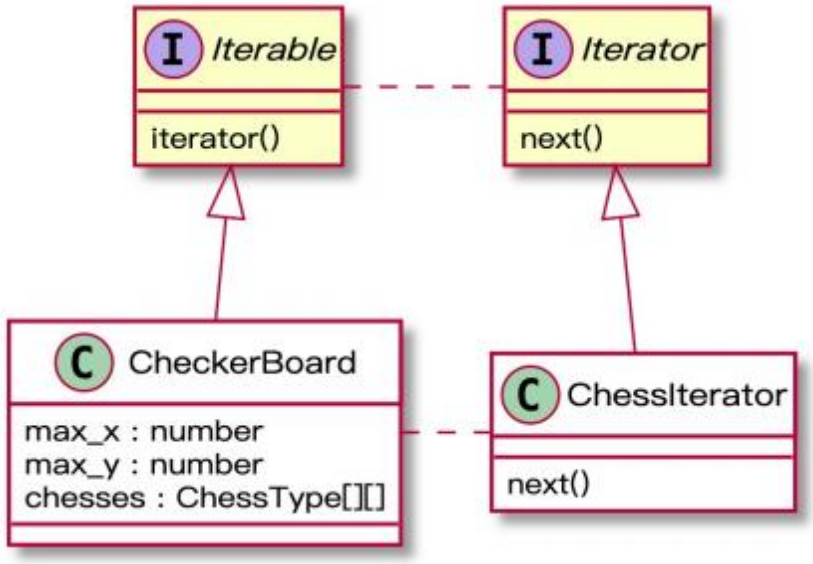
棋盘示例

- 使用迭代器模式后:

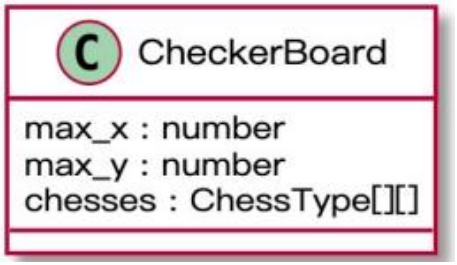
```
Iterator<Cell> cellIterator = checkerBoard.iterator();
while (cellIterator.hasNext())
    System.out.println(cellIterator.next());
```

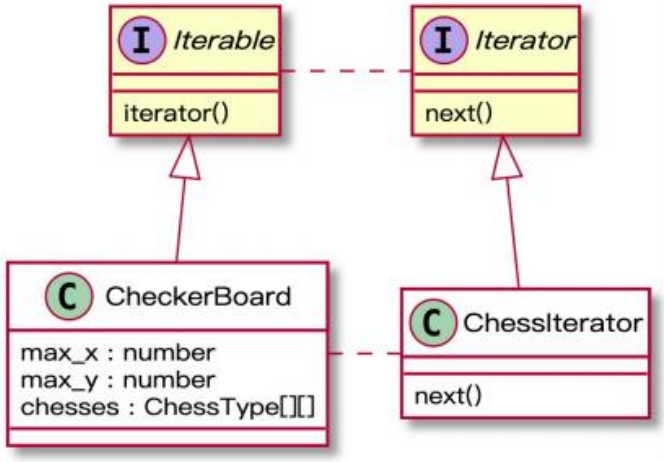
- 或者:

```
for (Cell cell : checkerBoard) {
    System.out.println(cell);
}
```



棋盘示例



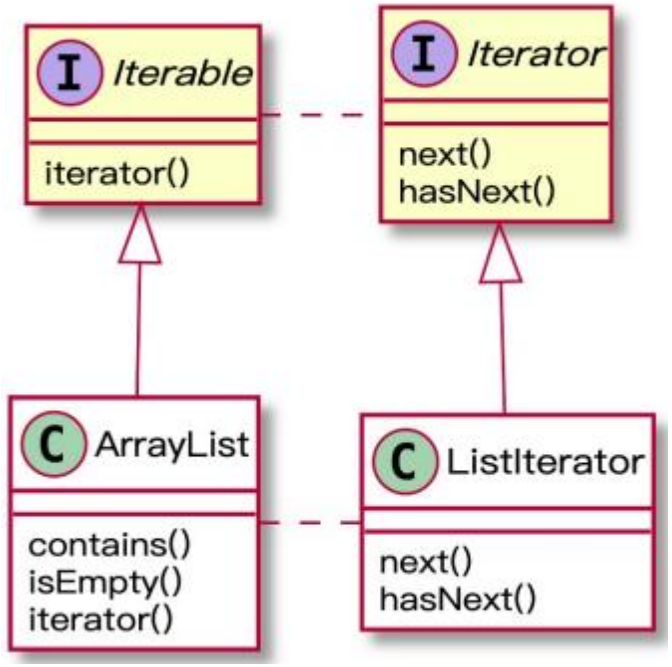



```
for (int y = 0; y < checkerBoard.getMax_y(); y++)
    for (int x = 0; x < checkerBoard.getMax_x(); x++)
        System.out.print(checkerBoard.getChess(x, y) + " ");
```

```
for (Cell cell : checkerBoard) {
    System.out.println(cell);
}
```

Java中的ArrayList

- ArrayList类可以提供一个ListIterator，他们与Iterable和Iterator之间具有如图所示的关系。




```
ArrayList<Integer> array = new ArrayList<Integer>();
for (Integer item : array) {
    System.out.print(item);
}
```

C++

- C++模板库中提供了地道的迭代器。但是在此之前，已经有了smartpointer的概念。
- 在运算符重载的章节中给出过一个smartpointer的例子：

```
int main() {
  //初始化一个容器
  const int sz = 10;
  Obj o[sz];
  ObjContainer oc;
  for(int i = 0; i < sz; i++)
    oc.add(&o[i]); // Fill it up

  //使用smartpointer遍历容器中的元素
  SmartPointer sp(oc); // Create an iterator
  do {
    sp->f(); // Pointer dereference operator call
    sp->g();
  } while(sp++);
} ///:~
```

语言对迭代器的支持

- C++

```
for(auto person : persons){
    for(auto item : person.shopping())
        result.push_back(item);
}
```

- Typescript

```
for(const person of persons){
    for(const item of person.shopping())
        result.push(item);
}
```

- Java

```
for(Person person : persons){
    for(Item item : person.shopping())
        result.add(item);
}
```

场景2：循环遍历

假设有一个从0到n-1的链表，逻辑上首位衔接，如果需要对列表从指定的位置进行循环遍历，应该如何设计。

```
public static <A> Iterator<A> rotated(List<A> list, int offset) {
    return new Iterator<A>() {

        private int index = 0;

        @Override
        public boolean hasNext() {
            return index < list.size();
        }

        @Override
        public A next() {
            A result = list.get((index + offset) % list.size());
            index++;
            return result;
        }
    };
}
```

场景2：循环遍历

```
Iterator<Integer> r = rotated(
        List.of(1, 2, 3, 4, 5, 6, 7),
        2
);
while (r.hasNext())
    System.out.println(r.next());

//output : 3 4 5 6 7 1 2
```

场景3：生成斐波那级数(scala)

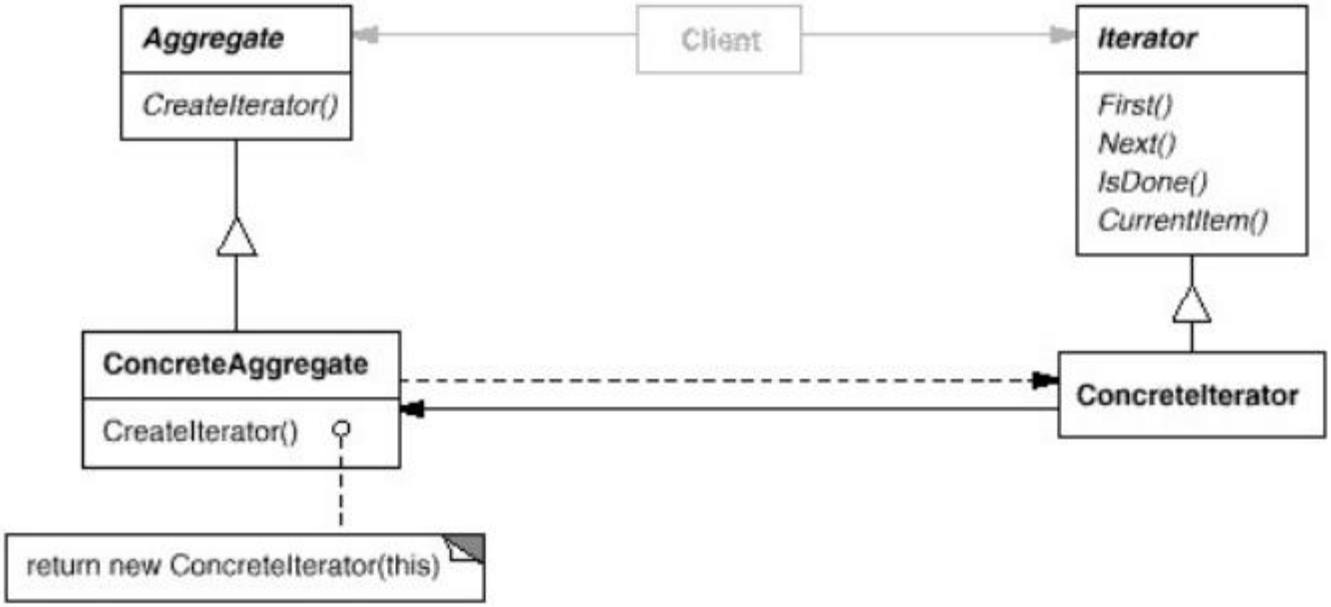
```
val fib = Iterator.iterate((0,1))(x => (x._2, x._1+x._2))
    .map(_._2)
```

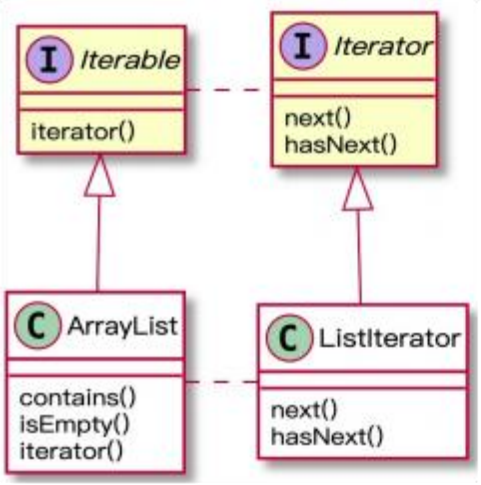
请考虑下面的问题：
1. 如何获取第一个大于3000的斐波那数
2. 如何获取前十个偶斐波那数

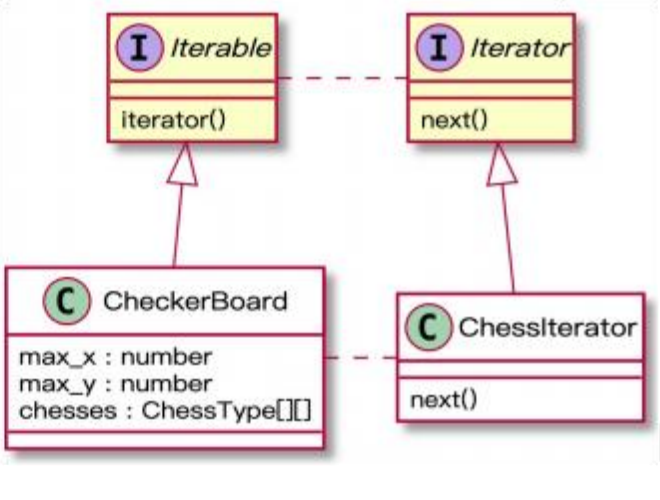
迭代器模式

- 聚合对象（如列表）应该对外提供一种访问其元素而不暴露其内部结构的方法。
- 此模式的关键思想将负责访问和遍历列表对象的职责分离出来，并将其放入迭代器对象中。.

结构


Aggregate
CreateIterator()
Client
Iterator
First()
Next()
IsDone()
CurrentItem()
ConcreteAggregate
CreateIterator()
ConcreteIterator
return new ConcreteIterator(this)


Iterable
iterator()
Iterator
next()
hasNext()
ArrayList
contains()
isEmpty()
iterator()
ListIterator
next()
hasNext()


Iterable
iterator()
Iterator
next()
CheckerBoard
max_x : number
max_y : number
chesses : ChessType[][]
ChessIterator
next()

参与者

迭代器（Iterator）：

- 迭代器定义了访问和遍历元素的接口。它提供了统一的方法来遍历聚合对象中的元素，而无需暴露聚合对象的内部表示。

- 迭代器通常需要包含至少两种种方法：hasNext() 用于检查序列中是否还有元素；next() 用于返回序列中的下一个元素；以及 remove()（可选）用于从聚合对象中移除迭代器最近返回的元素（这个操作是可选的，并非所有迭代器都需要支持）。

参与者

聚合对象（Aggregate）：

- 聚合对象是一个包含多个元素的容器，如列表、集合等。
- 它必须提供一个创建迭代器对象的方法，通常是 iterator() 方法，用于返回一个符合迭代器接口的对象，以便外部代码可以通过迭代器遍历聚合对象中的元素。
- 聚合对象可能还需要定义添加、删除、获取元素等操作，但这些操作与迭代器模式的核心职责（遍历元素）是分离的。

参与者

具体迭代器（Concrete Iterator）：

- 具体迭代器实现了迭代器接口，并包含了遍历聚合对象所需的状态。
- 它通过聚合对象的内部表示来遍历元素，但对外隐藏了这些内部细节。
- 具体迭代器可能需要根据聚合对象的具体实现来定制遍历逻辑。

参与者

客户端（Client）：

- 客户端代码通过迭代器来遍历聚合对象中的元素，而无需直接访问聚合对象的内部表示。
- 客户端使用迭代器提供的方法来遍历元素，并可以处理遍历过程中遇到的每个元素。

优点

- 单一职责原则。你可以通过将庞大的遍历算法提取到单独的类中，来清理客户端代码和集合。
- 开闭原则。你可以实现新的集合类型和迭代器，并将它们传递给现有代码，而不会破坏任何东西。
- 你可以并行遍历同一个集合，因为每个迭代器对象都包含它自己的迭代状态。
- 出于同样的原因，你可以延迟迭代并在需要时继续。

局限性

- 如果你的应用程序只处理简单的集合，那么应用这个模式可能会是过度设计。

  比如如果遍历棋盘并不频繁，对棋盘实现迭代器就是过度设计

- 使用迭代器可能比直接遍历某些特殊集合的元素效率更低。

思考题1

- 对于黑白棋的游戏，还有一个经常会用到的处理是，获取从一个落子的位置开始的，沿某个方向上的所有棋子(可能有八种方向)。
- 这种场景迭代器模式也能够适用?