```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.function.Function;

import Es7.Random;

public class Es1 {

    // 1.a
    public interface Predicate<T> extends Function<T, Boolean> {}

    // 1.b
    public interface Either<T> {
        T onSuccess(T x);
        void onFailure(T x) throws Exception;
    }

    // 1.c
    public static class SkippableArrayList<E> extends ArrayList<E> {
        public Iterator<E> iterator(Predicate<E> p, Either<E> f) {
            final Iterator<E> it = super.iterator();    // può anche essere un campo privato della c
            return new Iterator<E>() {
                @Override
                public boolean hasNext() {
                    return it.hasNext();
                }

                @Override
                public E next() {
                    E x = it.next();
                    if (p.apply(x))
                        return f.onSuccess(x);
                    else {
                        try {
                            f.onFailure(x);
                        }
                        catch (Exception e) {
                            e.printStackTrace();    // si può anche non fare niente dentro il catch,
                        }
                        return x;
                    }
                }
            };
        }
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        Collection<Integer> dst = new ArrayList<>();
        SkippableArrayList<Integer> src = new SkippableArrayList();

        Random r = new Random();

        Iterator<Integer> it = src.iterator((x) -> x > 5, new Either<Integer>() {
            @Override
            public Integer onSuccess(Integer x) {
                return x + 1;
            }

            @Override
            public void onFailure(Integer x) throws Exception {
                dst.add(x);
            }
        }
        );

        for(int i = 0; i < 10; i++)
            src.add(r.nextInt(par:10));

        while (it.hasNext())
            System.out.println(it.next());
    }

}
```

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.function.Function;

public class Es1 {

    // 1.a
    public static class FactorialThread extends Thread {
        private final int n;
        private long res;

        public FactorialThread(int n) {
            this.n = n;
        }

        @Override
        public void run() {
            res = fact(n);
        }

        public long getResult() {
            try {
                join();
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
            return res;
        }

        public int getN() {
            return n;
        }

        private static long fact(int n) {
            if (n <= 1) return 1;
            return n * fact(n - 1);
        }

    }

    // 1.b + 1.c
    public static List<FactorialThread> parallelFactorial(Iterable<Integer> c) {
        List<FactorialThread> r = new ArrayList<>();
        for (int n : c) {
            FactorialThread t = new FactorialThread(n);
            t.start();
            r.add(t);
        }
        return r;
    }

    // 1.d
    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        for (FactorialThread t : parallelFactorial(List.of(e1:0, e2:1, e3:2, e4:3, e5:11, e6:12, e7:23, e8:35))
            System.out.printf(format:"fact(%d) = %d\n", t.getN(), t.getResult());
    }

    // 1.e.i
    public static <A, B> List<B> map(Iterable<A> i, Function<A, B> f) {
        List<B> r = new ArrayList<>();
        for (A a : i)
            r.add(f.apply(a));
        return r;
    }

    // 1.e.ii
    public static Collection<FactorialThread> parallelFactorial2(Collection<Integer> c) {
        return map(c, (n) -> { FactorialThread t = new FactorialThread(n); t.start(); return t; });
    }
}
```

```java
package esami.esame2;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class FiboSequence implements Iterable<Integer>{

    private final int max;
    private final Map<Integer, Integer> cache = new HashMap<>();

    public FiboSequence(int max) {
        this.max = max;
    }

    @Override
    public Iterator<Integer> iterator() {
        return new Iterator<Integer>() {
            private int n = 0;

            @Override
            public boolean hasNext() {
                return n < max;
            }

            @Override
            public Integer next() {
                return fib(n++);
            }

            private int fib(int n) {
                if(n < 2) return 1;
                else {
                    Integer cached = cache.get(n);
                    if(cached != null) return cached;
                    else {
                        int result = fib(n - 1) + fib(n - 2);
                        cache.put(n, result);
                        return result;
                    }
                }
            }
        };
    }

    Run | Debug | Run main | Debug main
    public static void main(String[] args) {
        FiboSequence fs = new FiboSequence(max:10);
        for (int i : fs) {
            System.out.println(i);
        }
    }
}
```

```java
package esame2;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Comparator;
import java.util.Iterator;

public class BST<T> implements Iterable<T> {

    protected final Comparator<? super T> cmp;
    protected Node root;

    protected class Node {

        protected final T data;
        protected Node left, right;

        protected Node(T data, Node left, Node right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    public BST(Comparator<? super T> cmp) {
        this.cmp = cmp;
    }

    public void insert(T x) {
        root = insertRec(root, x);
    }

    protected Node insertRec(Node n, T x) {
        if (cmp.compare(n.data, x) < 0) {
            n.right = insertRec(n, x);
        }
        if (cmp.compare(n.data, x) > 0) {
            n.left = insertRec(n, x);
        }
        return new Node(x, left:null, right:null);
    }

    protected void dfsInOrder(Node n, Collection<T> out) {
        if (n != null) {
            if (n.left != null) {
                dfsInOrder(n.left, out);
                out.add(n.left.data);
            }
            out.add(n.data);
            if (n.right != null) {
                dfsInOrder(n.right, out);
                out.add(n.right.data);
            }
        }
    }

    @Override
    public Iterator<T> iterator() {
        Collection<T> out = new ArrayList<>();
        dfsInOrder(root, out);
        return out.iterator();
    }

    public T min() {
        Iterator<T> it = this.iterator();
        return it.next();
    }

    public T max() {
        Iterator<T> it = this.iterator();
        T last = it.next();
        while (it.hasNext()) {
            last = it.next();
        }
        return last;
    }
}
```

```cpp
#include <iostream>

using namespace std;

template <class A, class B>
class mypair
{
private:
    A first;
    B second;

public:
    mypair() : first(), second() {}

    mypair(A _first, B _second) : first(_first), second(_second) {}

    // copy constructor
    mypair(const pair<A, B> &p) : first(p.first), second(p.second) {}

    // assignment operator
    mypair<A, B> &operator=(const mypair<A, B> &p)
    {
        first = p.first;
        second = p.second;
        return *this;
    }

    // operator++
    mypair<A, B> operator++(int)
    {
        auto tmp = *this;
        first++;
        second++;

        return tmp;
    }

    A fst() const { return first; }
    B snd() const { return second; }

    mypair<A, B> operator+(const mypair<A, B>& p) const
    {
        return mypair<A, B>(first + p.first, second + p.second);
    }
};

int main()
{
    mypair<int, int> p1 = mypair<int, int>(1, 2);
    mypair<int, int> p2 = mypair<int, int>(3, 4);

    p1 = p2;
    p1++;
    cout << p1.fst() << " " << p1.snd() << endl;
    return 0;
}
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

template <typename T>
class matrix
{
public:
    using value_type = T;
    using iterator = typename vector<T>::iterator;
    using const_iterator = typename vector<T>::const_iterator;

    matrix() : rows(0), cols(0), scheme() {}
    matrix(int _row, int _cols, const T& data = T()) : rows(_row), cols(_cols) {
    scheme = vector<T>(rows * cols, data);
    }
    matrix(const matrix<T>& other) : rows(other.rows), cols(other.cols), scheme(other.scheme) {}

    matrix& operator=(const matrix<T>& other) {
    this->rows = other.rows;
    this->cols = other.cols;
    this->scheme = other.scheme;

    return *this;
    }

    const T& operator()(int i, int j) const {
    return scheme[ i * cols + j];
    }

    T& operator()(int i, int j) {
    return scheme[ i * cols + j];
    }

    void print() const {
        for (size_t i = 0; i < rows; ++i) {
            for (size_t j = 0; j < cols; ++j) {
                cout << scheme[i * cols + j] << " ";
            }
            cout << endl;
        }
    }

    iterator begin() {
        return scheme.begin();
    }

    iterator end() {
        return scheme.end();
    }

    const_iterator begin() const {
        return scheme.begin();
    }

    const_iterator end() const {
        return scheme.end();
    }

private:
    int rows;
    int cols;
    vector<T> scheme;
};

int main()
{
    matrix<int> x(5, 6, 34);
    x(2, 3) = 100;
    for(typename matrix<int>::iterator i = x.begin(); i != x.end(); i++){
    cout << *i << " ";
    }
    cout << endl;
    x.print();
    int a = x(2, 3);
    cout << a << endl;
    return 0;
}
```