OBJECT-ORIENTED SYSTEMS DESIGN

[Exercise]: Generics and the ArrayList Class

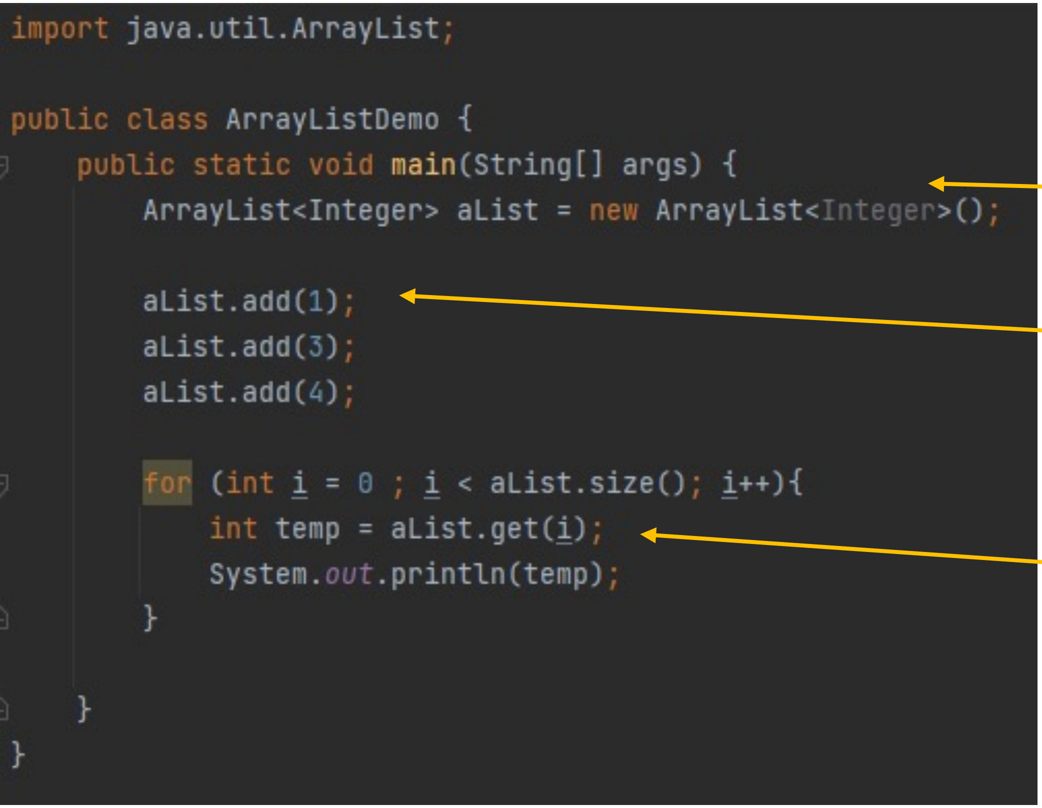
2022/06/08

Department of Computer Science,
Hanyang University

Lecturer: Taeuk Kim
TA: Taejun Yoon, Seong Hoon Lim, Young Hyun Yoo

The ArrayList Class

- **An ArrayList serves the same purpose as an array, except that an ArrayList can change length while the program is running.**



```
import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args) {
        ArrayList<Integer> aList = new ArrayList<Integer>();

        aList.add(1);
        aList.add(3);
        aList.add(4);

        for (int i = 0 ; i < aList.size(); i++){
            int temp = aList.get(i);
            System.out.println(temp);
        }

    }
}
```

The **base type** of an ArrayList must be a **class type.**

The add method is used to set an element for the first time in an ArrayList.

We should use get() method to get an item from an ArrayList.

The `ArrayList` Class

- To insert items into the `ArrayList` for the first time you can use the `add()` method.

```
aList.add(“Goodbye”);
aList.add(“world”);
```

Goodbye	world				
0	1	2	3	4	...

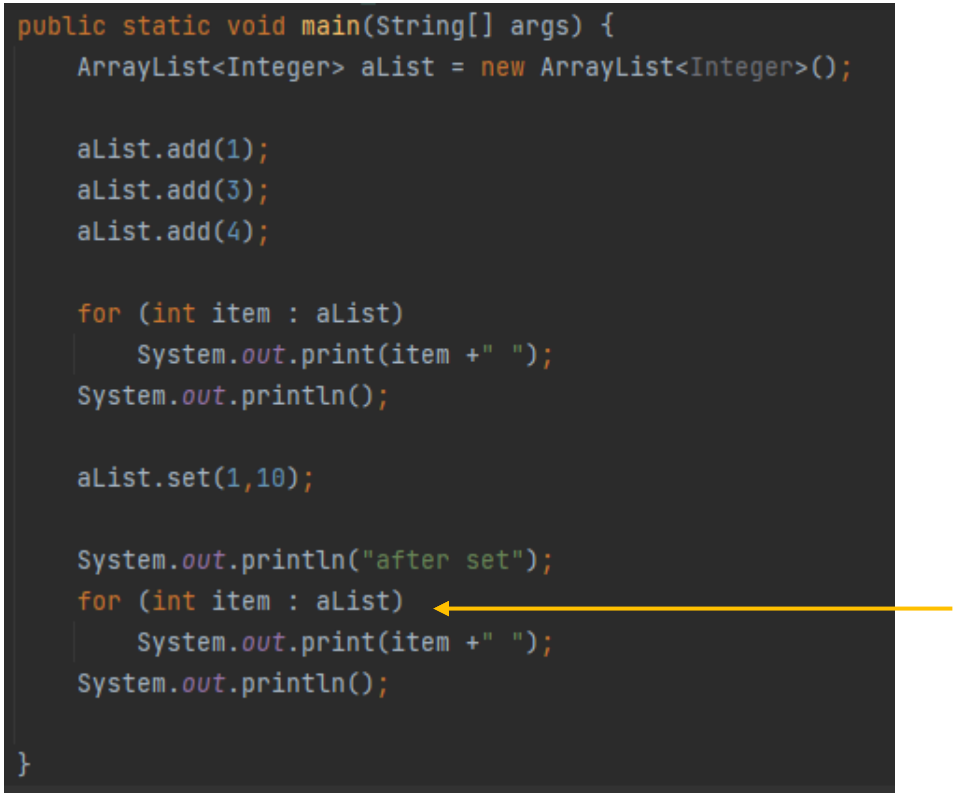
- The `add()` method is overloaded and can accept another parameter: `add(index, object).`

```
aList.add(1,“cruel.”);
```

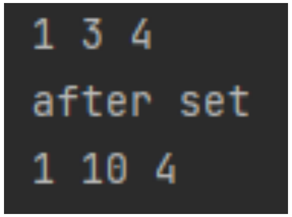
Goodbye	cruel	world			
0	1	2	3	4	...

The `ArrayList` Class

- **The `set` method is used to replace any existing element, and the `get` method is used to access the value of any existing element.**



```
public static void main(String[] args) {
    ArrayList<Integer> aList = new ArrayList<Integer>();

    aList.add(1);
    aList.add(3);
    aList.add(4);

    for (int item : aList)
        System.out.print(item +" ");
    System.out.println();

    aList.set(1,10);

    System.out.println("after set");
    for (int item : aList)
        System.out.print(item +" ");
    System.out.println();

}
```



```
1 3 4
after set
1 10 4
```

`set` can **only** reset an element at an index that already contains an element.

As with arrays, the **for-each loop** can be used to cycle through (*iterate*) all the elements in an collection (like an **ArrayList**).

Generics

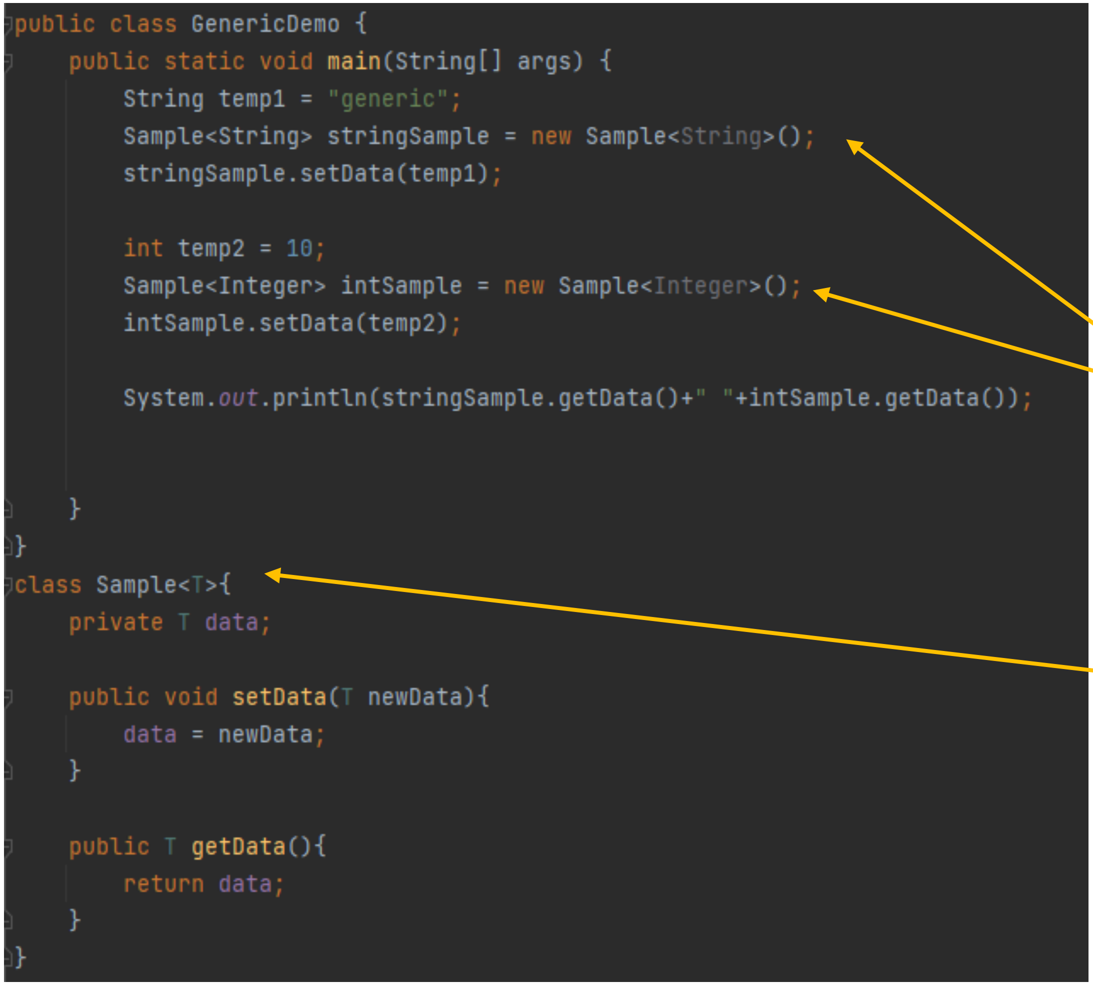
- **Classes and methods can have a type parameter.**
  - A type parameter can have any reference type (i.e., any class type) plugged in for the type parameter.
  - When a specific type is plugged in, this produces a specific class type or method.
  - Traditionally, a single uppercase letter (T) is used for a type parameter, but any non-keyword identifier may be used.

- **A class definition with a type parameter is stored in a file and compiled just like any other class.**
  - Once a parameterized class is compiled, it can be used like any other class.
  - However, the class type plugged in for the type parameter must be specified before it can be used in a program.
  - Doing this is said to *instantiate* the generic class.

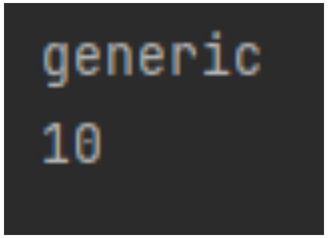
```
Sample<String> object = new Sample<String>();
```

Generics

- **Example**



```
public class GenericDemo {
    public static void main(String[] args) {
        String temp1 = "generic";
        Sample<String> stringSample = new Sample<String>();
        stringSample.setData(temp1);

        int temp2 = 10;
        Sample<Integer> intSample = new Sample<Integer>();
        intSample.setData(temp2);

        System.out.println(stringSample.getData()+" "+intSample.getData());


    }
}
class Sample<T>{
    private T data;

    public void setData(T newData){
        data = newData;
    }

    public T getData(){
        return data;
    }
}
```



```
generic
10
```

Type parameter **T** can be any class type such as **String**, **Integer….**

Mark **<T>** when defining generics.

OBJECT-ORIENTED SYSTEMS DESIGN

[Exercise]: Collections, Maps, and Iterators

2022/06/08

Department of Computer Science,
Hanyang University

Lecturer: Taeuk Kim
TA: Taejun Yoon, Seong Hoon Lim, Young Hyun Yoo

Collections

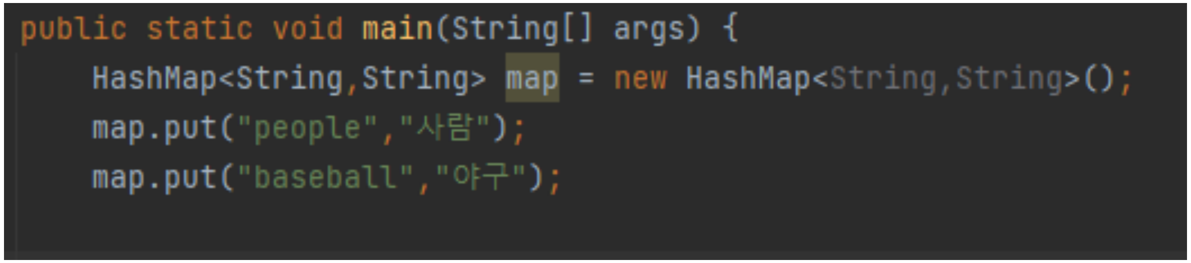
- **A Java collection is any class that holds objects and implements the `Collection` interface.**
  - For example, the `ArrayList<T>` class is a Java collection class, and implements all the methods in the `Collection` interface.
  - Collections are used along with *iterators*.
- **The `Collection` interface is the highest level of Java's framework for collection classes.**
  - All of the collection classes discussed here can be found in package `java.util.`

Maps

- **The Java *map* framework deals with collections of ordered pairs.**
  - For example, a key and an associated value.

- **Objects in the map framework can implement mathematical functions and relations, so can be used to construct database classes.**

- **HashMap<K,V> Class**

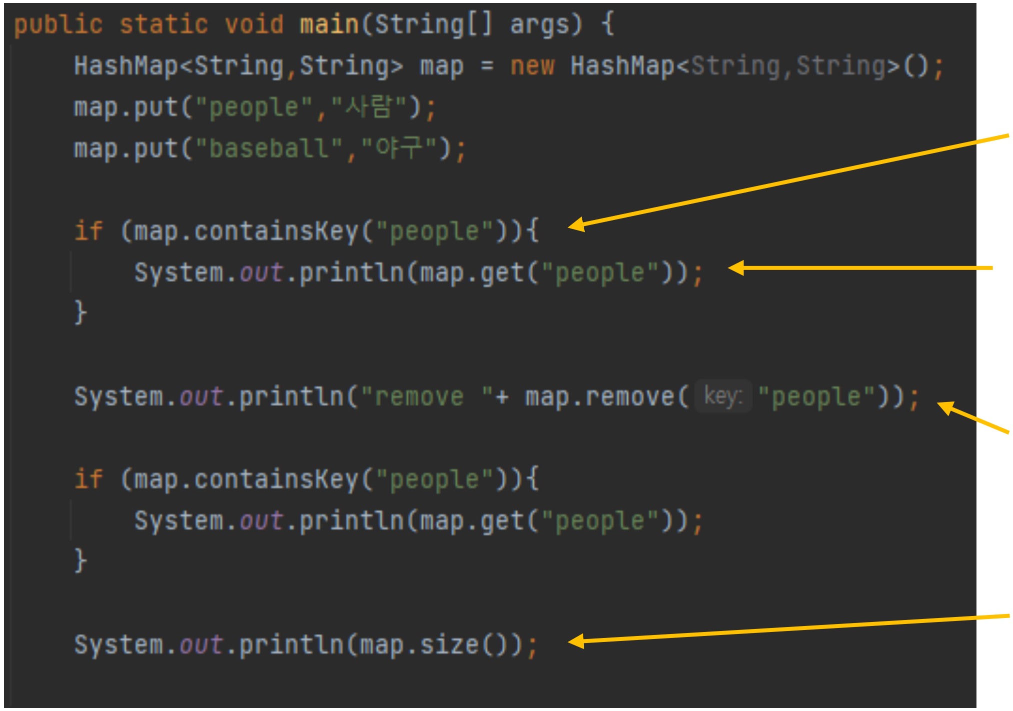


```
public static void main(String[] args) {
    HashMap<String,String> map = new HashMap<String,String>();
    map.put("people","사람");
    map.put("baseball","야구");
```

key	value
People	사람
baseball	야구

HashMap<K,V> Class

- **Example**



```
public static void main(String[] args) {
    HashMap<String,String> map = new HashMap<String,String>();
    map.put("people","사람");
    map.put("baseball","야구");

    if (map.containsKey("people")){
        System.out.println(map.get("people"));
    }

    System.out.println("remove "+ map.remove( key: "people"));

    if (map.containsKey("people")){
        System.out.println(map.get("people"));
    }

    System.out.println(map.size());
```

The **`containsKey`** method return **`true`** if **`HashMap`** has the corresponding key.

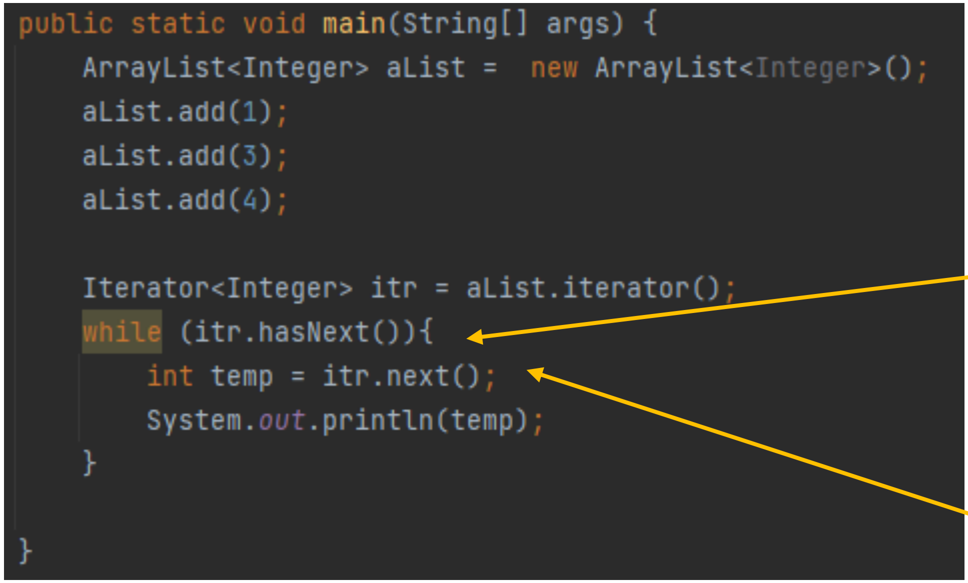
The **`get`** method return the value for the key.

The **`remove`** method removes the input key and corresponding value after returning the value.

`HashMap` has a method **`size()`** that returns its size.

Iterators

- **An iterator is an object that is used with a collection to provide sequential access to the collection elements.**

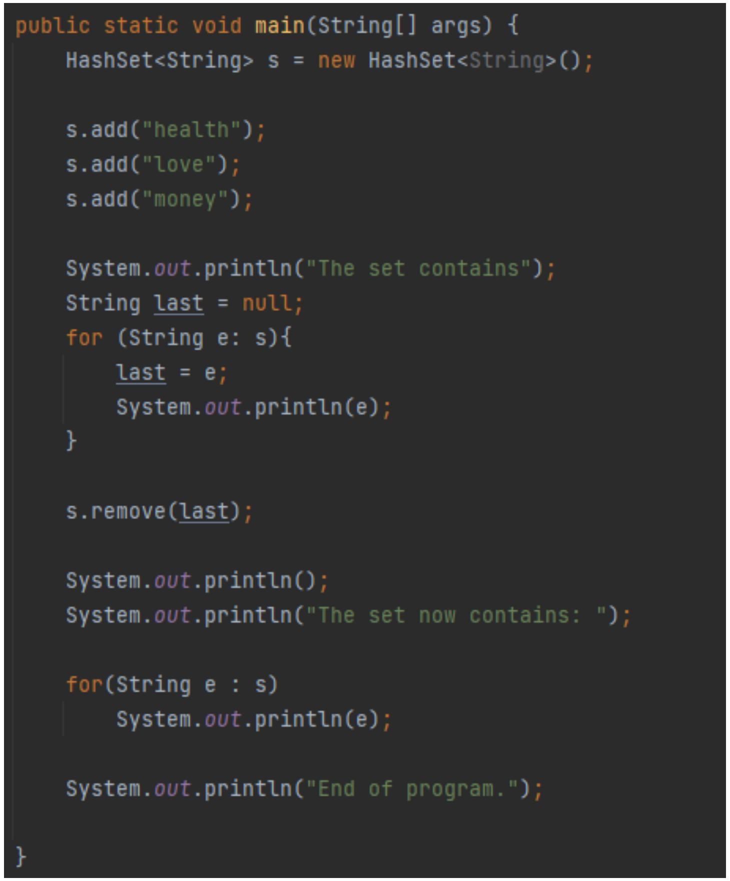


```
public static void main(String[] args) {
    ArrayList<Integer> aList =  new ArrayList<Integer>();
    aList.add(1);
    aList.add(3);
    aList.add(4);

    Iterator<Integer> itr = aList.iterator();
    while (itr.hasNext()){
        int temp = itr.next();
        System.out.println(temp);
    }

}
```

`hasNext()` returns **true** if `next()` has not yet returned all the elements in the collection; return **false** otherwise.

`next()` method returns the next element of the collection that produced the iterator.

For-Each Loops as Iterators

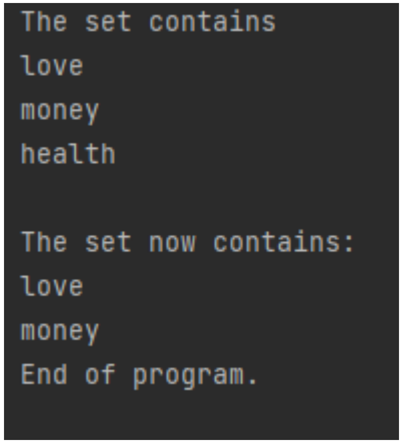


```
public static void main(String[] args) {
    HashSet<String> s = new HashSet<String>();

    s.add("health");
    s.add("love");
    s.add("money");

    System.out.println("The set contains");
    String last = null;
    for (String e: s){
        last = e;
        System.out.println(e);
    }

    s.remove(last);

    System.out.println();
    System.out.println("The set now contains: ");

    for(String e : s)
        System.out.println(e);

    System.out.println("End of program.");

}
```

- **Although it is not an iterator, a for-each loop can serve the same purpose as an iterator.**
  - A for-each loop can be used to cycle through each element in a collection.

- **For-each loops can be used with any of the collections discussed here.**

output



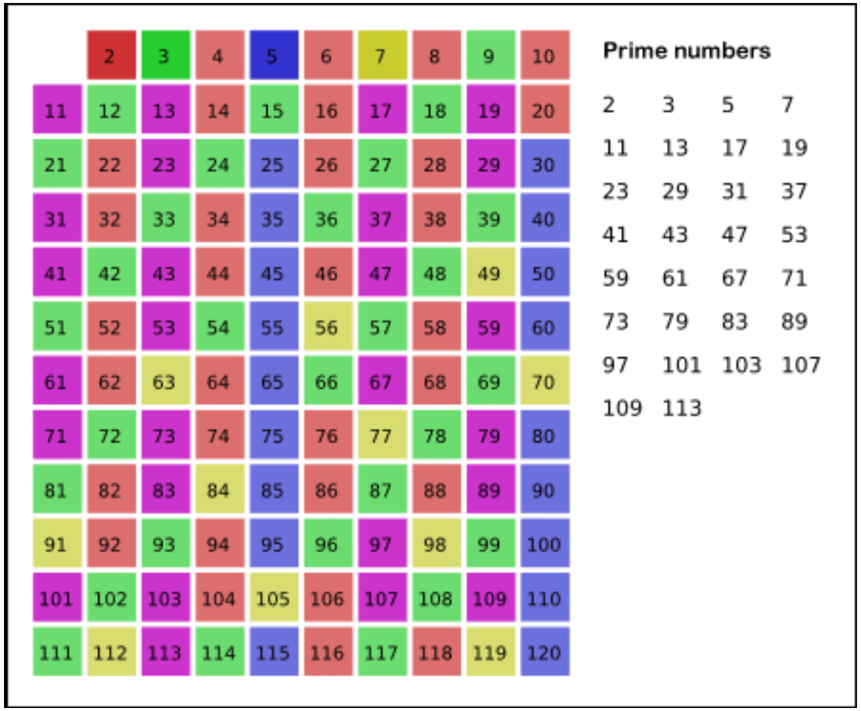
```
The set contains
love
money
health

The set now contains:
love
money
End of program.
```

Practice

/WeekN/Eratos.java
/Main.java

The Sieve of Eratosthenes

- **The Sieve of Eratosthenes**
  - It is an algorithm that generates **prime numbers.**
  - First remove 1 in the list.
  - Then remove the multiples of 2 and 2 and add 2 to the prime number list.
  - Remove the multiples of 3 and 3, add 3 to the prime number list.
  - Do the same process iteratively with the next remaining number.



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

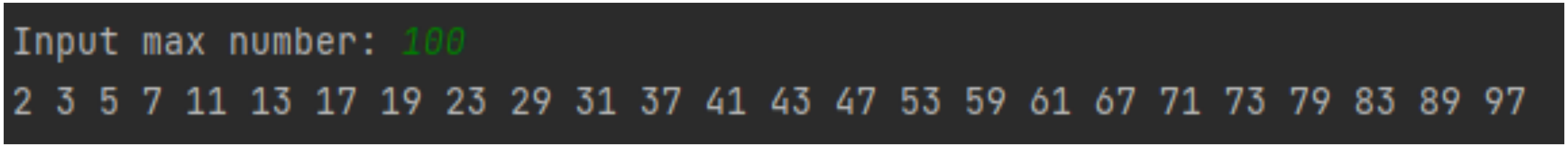
Prime numbers

2 3 5 7
11 13 17 19
23 29 31 37
41 43 47 53
59 61 67 71
73 79 83 89
97 101 103 107
109 113

Practice

- **Eratos.java**
  - Implement the **Sieve of Eratosthenes** algorithm using an **ArrayList** of **Integers.**
  - The **Eratos** class has a static method **sieve** with a parameter integer **n** that returns an **ArrayList** that contains prime numbers less than **n.**

- **Main.java**
  - Print out prime numbers less than **n** (given by a user) using **Iterator**.



```
Input max number: 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Time for Practice

Get it started, and ask TAs if you are in a trouble.