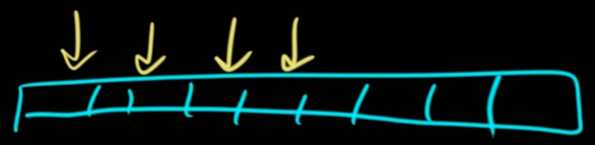
```java
int[] a = {10, 20, 30};
for (int i = 0; i < a.length; i++) {
  System.out.println(a[i]);
}

for (int x : a) {
  System.out.printf("element: %d\n", x);
}
```



```java
ArrayList<String> l1 = new ArrayList<>();
l1.add("Tesla");
l1.add("Jeep");
l1.add("Mazda");
for (String x : l1) {
  System.out.println(x);
}
```

.size()

```java
Set<Double> l2 = new HashSet<>();
l2.add(Math.PI);
l2.add(Math.E);
for (double x : l2) {
  System.out.println(x);
}
```

.size()

1+
0+

```java
Iterator<String> it = l1.iterator();
while (it.hasNext()) {
  String x = it.next();
  System.out.println(x);
}

for (Iterator<String> it0 = l1.iterator(); it0.hasNext(); ) {
  String x = it0.next();
  System.out.println(x);
}
```

must have

Iterator → Iterable

```java
for (String x : l1) {
  System.out.println(x);
}
```

Iterator<A> interface

hasNext() → boolean

next() → A

```java
private static final String[] data = {
  "January",
  "February",
  "March",
  "April",
  "May",
  "June",
  "July",
  "August",
  "September",
  "October",
  "November",
  "December",
};
```

```java
return new Iterator<>() {
  int index = 0;

  @Override
  public boolean hasNext() {
    return index < data.length;
  }

  @Override
  public String next() {
    String x = data[index];
    index++;
    return x;
  }
};
```

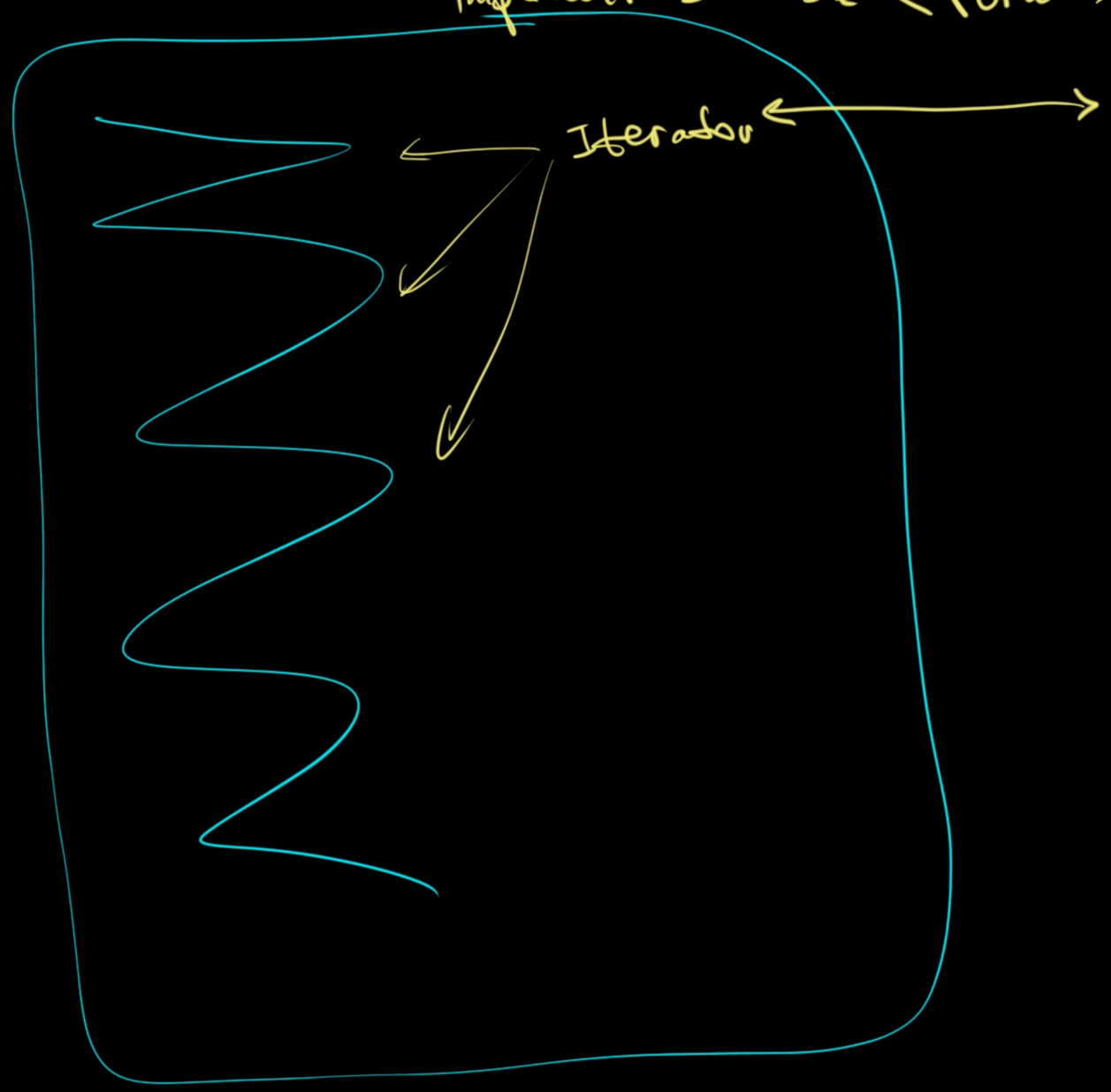
Iterable <A>

for (A a : aa) {

}

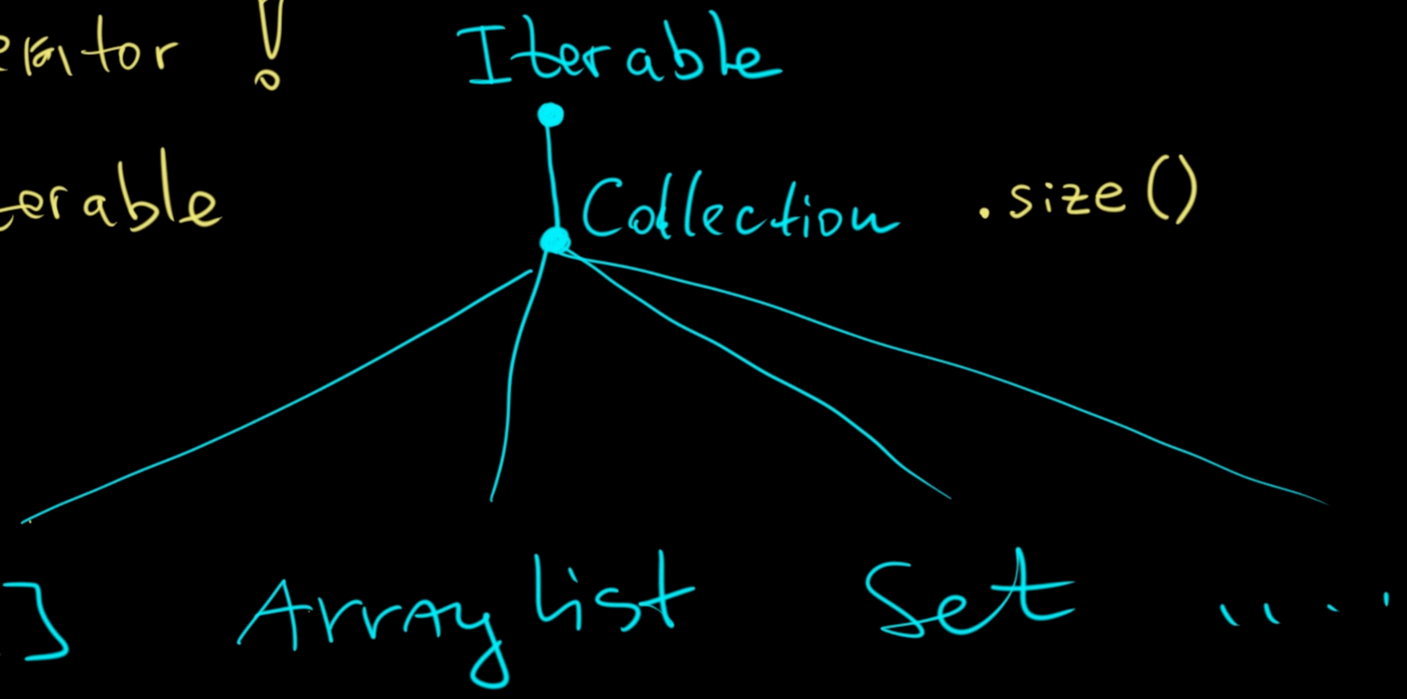
Iterator <A>
(index,
next
hasNext)

implement Iterable <Point>



Iterator !

Iterable



$x > y$

$x == y$

$x < y$

int

$f: (A\ a1, A\ a2) \rightarrow ???$

$=$

$\neq$ $<$ $>$

A

$a1 > a2$

$a1 == a2$

$a1 < a2$

$f: (A\ a1, A\ a2) : int$

$\ldots < 0$ $\quad a1 < a2 \quad -1$

0 $\quad a1 == a2 \quad 0$

$\ldots > 0$ $\quad a1 > a2 \quad 1$

$\boxed{x1 - x2}$

$f: (int\ x1, int\ x2)$

if $x1 < x2$ $\quad -1$

$x1 > x2$ $\quad +1$

else $\quad 0$

```java
public void sort(Comparator<E> c)
```

```java
public interface Comparator<T>
    int compare(T o1, T o2);
```

```java
pizzas.sort(comparatorBySize);
pizzas.forEach(System.out::println);

System.out.println("-------");
pizzas.sort(comparatorByPrice);
pizzas.forEach(System.out::println);
```

```
Pizza[size=30, name=A, price=3.5]
Pizza[size=60, name=A, price=5.9]
Pizza[size=60, name=B, price=4.9]
-------
Pizza[size=30, name=A, price=3.5]
Pizza[size=60, name=B, price=4.9]
Pizza[size=60, name=A, price=5.9]
```

```
Pizza[size=30, name=A, price=3.5]
Pizza[size=60, name=B, price=4.9]
Pizza[size=60, name=A, price=5.9]
```

0,0000001
9000002

1. xxxx

Pizza
- name : string
- size : int
- weight : int
- price : int
- extra : []string

→ size
→ weight
→ price
→ name (alphabet)
→ extra.length
→ size + price + name + extra

$f : (A, A) : int$

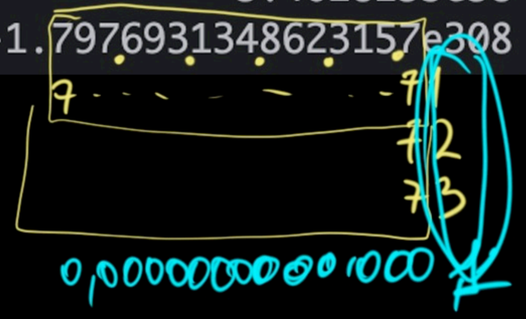
-1	<0
0	=0
+1	>0

$A < A$

$A == A$

$A > A$

```
f32:                          -3.4028235e38
f64:                  -1.7976931348623157e308
```



```java
Comparator<Pizza> comparatorBySizePrice = new Comparator<>() {
  @Override
  public int compare(Pizza o1, Pizza o2) {
    int s = o1.size - o2.size;
    if (s != 0) return s;
    return (int) (o1.price - o2.price);
  }
};
```

Comparator<A> ≡ compare(A a1, A a2) → int

<0 a1<a2
=0 a1=a2
>0 a1>a2

Comparable<A>

compare(this, o)

```java
public interface Comparable<T>
  int compareTo( @NotNull T o);
```

→ int <0 =0 >0

```java
record Pizza(int size, String name, double price) implements Comparable<Pizza> {
  @Override
  public int compareTo(Pizza that) {
    return this.size - that.size;
  }
}
```

Comparable - 1

Int
long
char

Comparator - 0+

Pizza
User
ShoppingCart

totalPrice
numberItems

ArrayList

```
void sort(Comparator<? super E> c)
```

```
pizzas.sort(comparatorBySize);
```

many ways to sort

```
Collections.sort(pizzas);
```

only one way to sort

```
static <T extends Comparable<T>> void sort( @NotNull List<T> list)
```

```java
record Box(int x) implements Comparable<Box> {
  @Override
  public int compareTo(Box that) {
    return compBox.compare(this, that);
  }
}

static Comparator<Box> compBox = new Comparator<>() {
  @Override
  public int compare(Box o1, Box o2) {
    return o1.x - o2.x;
  }
};
```

Comparator => Comparable

```java
record Box(int x) implements Comparable<Box> {
  @Override
  public int compareTo(Box that)
}

Comparator<Box> compBox2 = new Comparator<>() {
  @Override
  public int compare(Box o1, Box o2) {
    return o1.compareTo(o2);
  }
};
```

Comparable => Comparator

Comparator.compare (a1, a2) -> int 0+

Comparable this (that) -> int
linked to class =1

a1 < a2 <0, -1
a1 = a2 0
a1 > a2 >0, +1