int min(int[])

```
int[] collect(int[] xs, FilterFn fn)
```

fn.test(...)

```
interface FilterFn {
  boolean test(int x);
}
```

=!

```
FilterFn f1c = x -> {
  return x < 0;
};
```

```
FilterFn f1d = x ->
  x < 0;
```

```
var f1a = new FilterFn() {
  @Override
  public boolean test(int x) {
    return x < 0;
  }
};
```

```
FilterFn f1e = x -> x < 0;
```

```
int[] as2 = collect(as, f1a);
int[] as3 = collect(as, x -> x > 0);
```

```
FilterFn f1b = (int x) -> {
  return x < 0;
};
```

```
IntStream.generate(() -> (int) (Math.random() * 101 - 50))
```

interface X {
int get()
}

```
IntStream.generate(new IntSupplier() {
  @Override
  public int getAsInt() {
    return (int) (Math.random() * 101 - 50);
  }
})
```

```
IsBetween isBetween = x -> x >= s && x <= t;
```

boolean

interface Y {
bool test(int x)
}

```
interface IsBetween extends Function<Integer, Boolean>
```

```
public interface Function<T, R>
```

```
R apply(T t);
```

T = Int
R = Bool

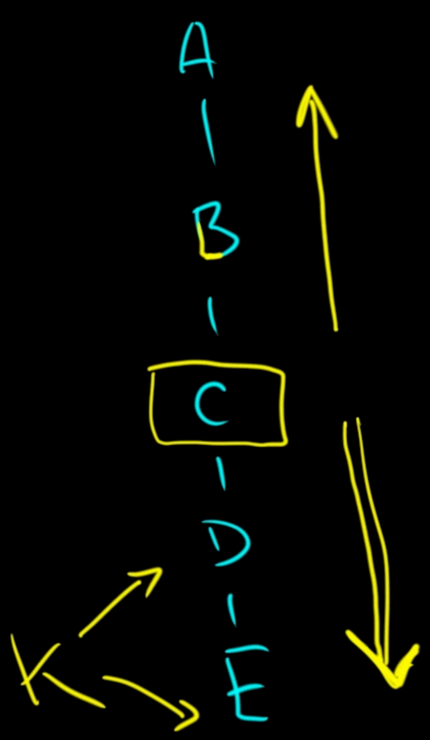
```
m.forEach((k, v) ->
  System.out.printf("%s %s\n", k, v)
);
```

interface Z {
void eat(k : K, v : V)
}

Map<K, V>

```
BiConsumer<? super K, ? super V> action
```

```
BiConsumer<T, U>
void accept(T t, U u);
```



? super C

K extends C


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

<T>

void sort (List <T> l)

? super X

K extends Y

f(x) → Y

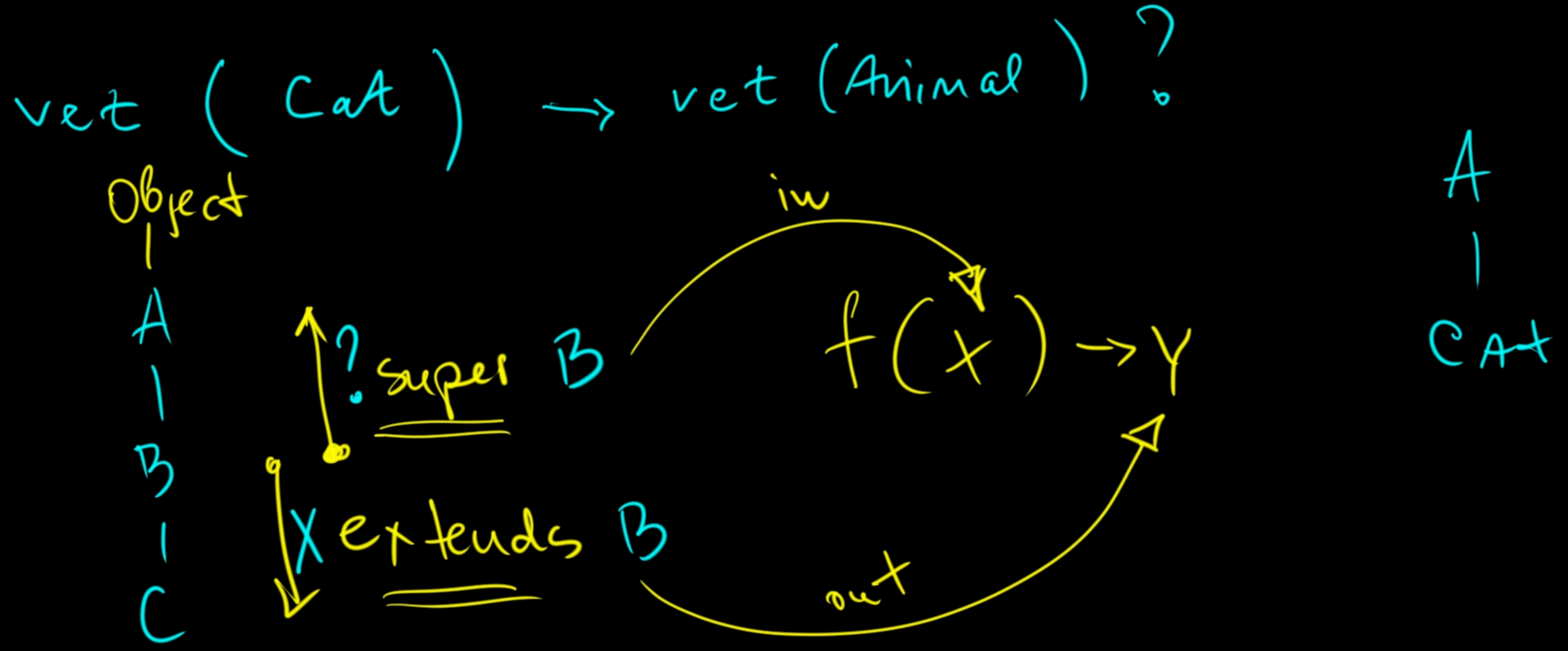
whatever(): K extends Animal

return new Cat

return new Dog

K extends Animal

<K extends Animal> K whatever() { ... }


vet (Cat) → vet (Animal) ?
Object
A
B
C
? super B
X extends B
in
out
f(x) → Y

A
Cat

Stream API ≈ Iterator<A>

Box<A> (1)

[...]

f(↓)

.forEach

1. initiation

```
Stream.generate(() -> (int) (Math.random() * 101 - 50))
```

```
int[] xxs = {1,2,3,4,5};
Arrays.stream(xxs);
```

```
pizzas.stream()
```

Stream<A> (0+1)

2. operations
- .map (f: A → B) : Stream<B>
- .filter (f: A → Boolean) : Stream<A>

3. termination
- .collect
- .forEach
- .count

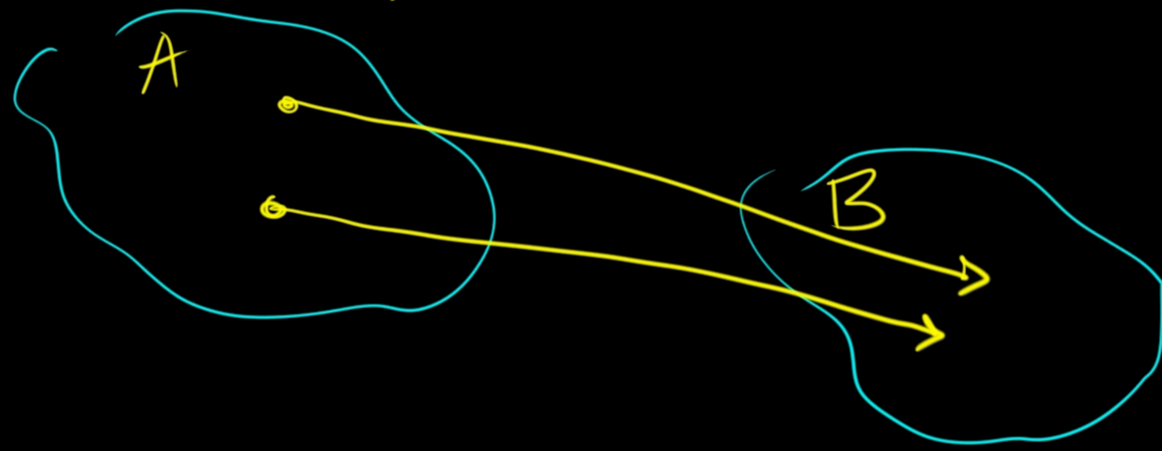
```java
for (int x: xs) {
  System.out.println(x);
}

xs.stream()
  .forEach(x -> System.out.println(x));
```

$x \to x+1$

$x \to x \% 10$

$f: A \to B$




```java
for (int x: xs) {
  int y = x + 1;
  System.out.println(y);
}

xs.stream()  Stream<Integer>
  .map(x -> String.format("%d^2 = %d", x, x*x))
  .forEach(x -> System.out.println(x));
```

filter(f: A -> Boolean)

A → A
↘ x

[1,2,3]
[1,3]

flatMap(f: A -> Stream<B>)

```
List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) List<Integer>
   .stream() Stream<Integer>
   .flatMap(x -> Stream.of(-x, x))
   .collect(Collectors.toUnmodifiableList());
```

```
[-1, 1, -2, 2, -3, 3, -4, 4, -5, 5, -6, 6, -7, 7, -8, 8, -9, 9, -10, 10]
```

```
Stream<Stream<Integer>> xxs =
    List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  List<Integer>
        .stream()  Stream<Integer>
        .map(x -> Stream.of(-x, x));
```

[[-1,1],[-2,2],[-3,3],...]

```
Stream<Integer> list =
    List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  List<Integer>
        .stream()  Stream<Integer>
        .flatMap(x -> Stream.of(-x, x));
```

[-1,1,-2,2,-3,3 ....]

.flatMap(x -> Stream.of(f(x))) ≡ .map(x -> f(x))

.flatMap(x -> if f(x) Stream.of(x) else Stream.empty) ≡ .filter(x -> f(x))

1. init

coll.stream()
Arrays.stream(T[])
Stream.generate
.....

} Stream<T>

---

2. operations

.map 1 → 1
.filter N → M M ≤ N
.flatMap N → M
.distinct
.limit(10)

1000.000
.limit(10)

---

3. termination

.forEach(f) → void
.collect(toList
toSet
to.....) → List<T>
Set<T>
....<T>

.count
.allMatch
noneMatch
anyMatch

```java
static boolean predicatePerson(Person p) {
  return p.age >= 18;
}

public static void main(String[] args) {

  List<Person> people = List.of(
    new Person(name:"Jim", age:20),
    new Person(name:"Tim", age:22),
    new Person(name:"Ben", age:32)
  );

  // f: person -> boolean
  Predicate<Person> pp = p -> p.age >= 18;

  boolean all = people.stream()
    .allMatch(pp);

  boolean any = people.stream()
    .anyMatch(Match::predicatePerson);
```

Person -> Boolean

method reference

```java
boolean any = people.stream()
  .anyMatch(Match::predicatePerson);

boolean any1 = people.stream()
  .anyMatch(p -> predicatePerson(p));
```

```java
boolean none = people.stream()
  .noneMatch(p -> p.age >= 18);
```

heap

person -> boolean

PP = new Predicate<Person> override

test (P: -)
p age >18

```java
public static void countApplesAndOranges(
  int s, int t, int a, int b,
  List<Integer> apples,
  List<Integer> oranges) {

  BiFunction<Integer, List<Integer>, Long> counter =
    (center, distances) -> distances.stream()
        .map(d -> center + d)
        .filter(x -> x >= s && x <= t)
        .count();

  long apple_count = counter.apply(a, apples);
  long orange_count = counter.apply(b, oranges);
  System.out.printf("%d\n%d\n", apple_count, orange_count);
}
```

```java
public static void countApplesAndOranges(
  int s, int t, int a, int b,
  List<Integer> apples,
  List<Integer> oranges) {

  int apple_count = 0;
  for (int distance : apples) {
    int pos = a + distance;
    if (pos >= s && pos <= t) {
      apple_count++;
    }
  }

  int orange_count = 0;
  for (int distance : oranges) {
    int pos = b + distance;
    if (pos >= s && pos <= t) {
      orange_count++;
    }
  }

  System.out.printf("%d\n%d\n", apple_count, orange_count);
}
```

```java
static String cleanup(String s) {
  return s.chars() IntStream
    .mapToObj(c -> (char) c) Stream<Character>
    .filter(Character::isLetter)
    .map(Object::toString) Stream<String>
    .collect(Collectors.joining()) String
    .trim();
}
```

```java
HashMap<String, Integer> m = new HashMap<>();

String[] ss = text.split( regex: " ");

for (String s : ss) {
  String s2 = cleanup(s);
  if (s2.isEmpty()) continue;
  m.merge(s2, value: 1, (a, b) -> a + b);
}

m.forEach((k, v) ->
  System.out.printf("%-20s %s\n", k, v)
);
```

2 1

```java
Arrays.stream(text.split( regex: " ")) Stream<String>
  .map(MapApp1::cleanup) 3
  .filter(s -> !s.isEmpty()) 4
  .collect(Collectors.groupingBy(s -> s, Collectors.counting())) Map<String, Long>
  .forEach((k, v) ->
    System.out.printf("%-20s %s\n", k, v)
  );
```

5