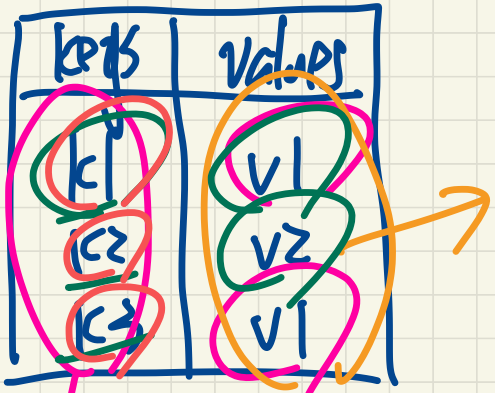
Test 2 Review

Monday November 25

Implement a Map



may have duplicates

unique

map(k1)

map(k3)

v1

v1

size: N

↳ search: O(N)

scale

↳ size of map : 1M

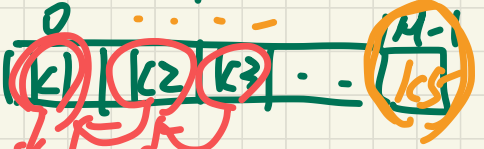
Strategy 1

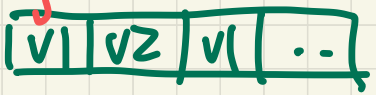
search(k1)

search(value)

keys

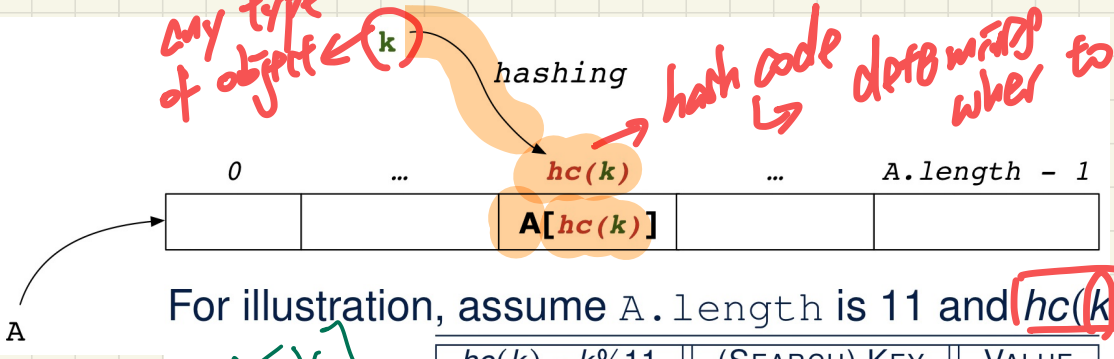
values





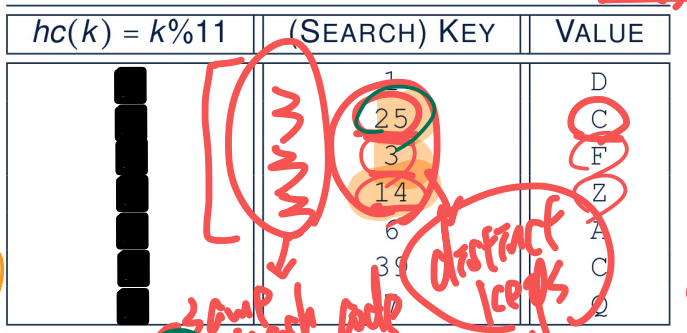
Strategy 2 ..

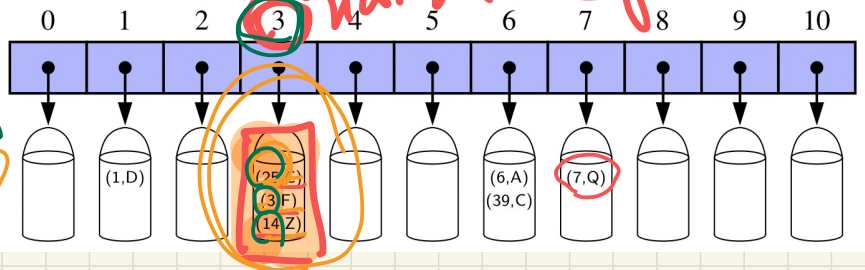
Implementing a Hash Table via Hashing



- Converting k to hc(k)
- Indexing into A[hc(k)]

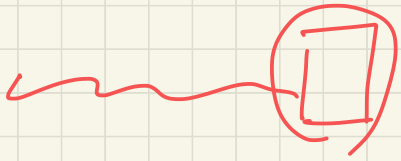
For illustration, assume `A.length` is 11 and hc(k) = k%11



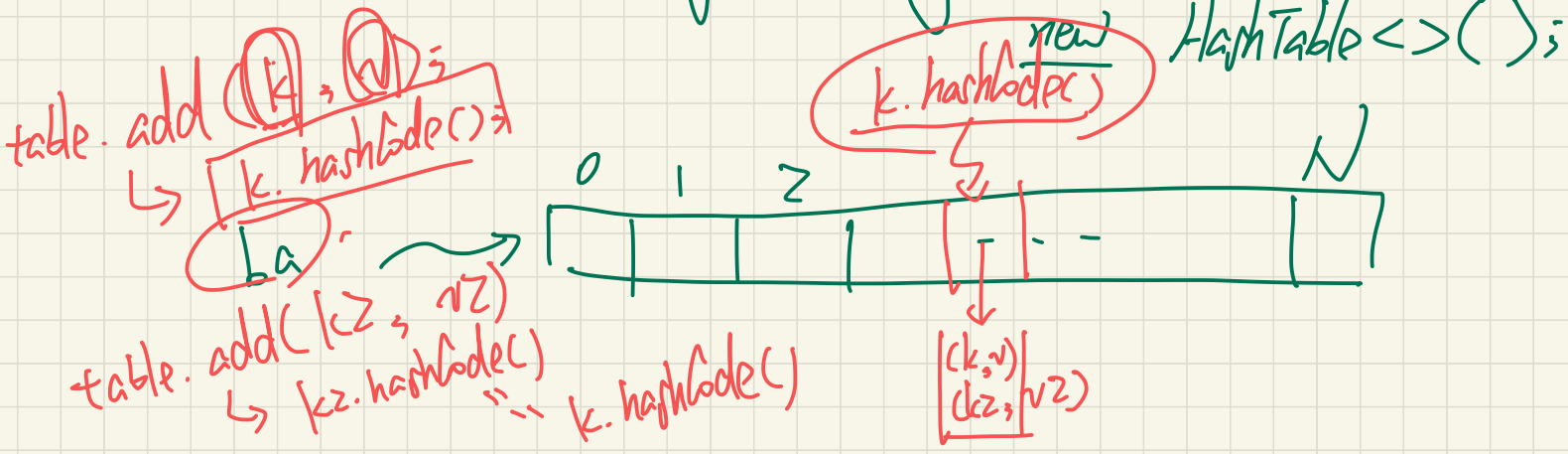


ArrayList<..> list

list.add(..)



HashTable<Integer, String> table = new HashTable<>();

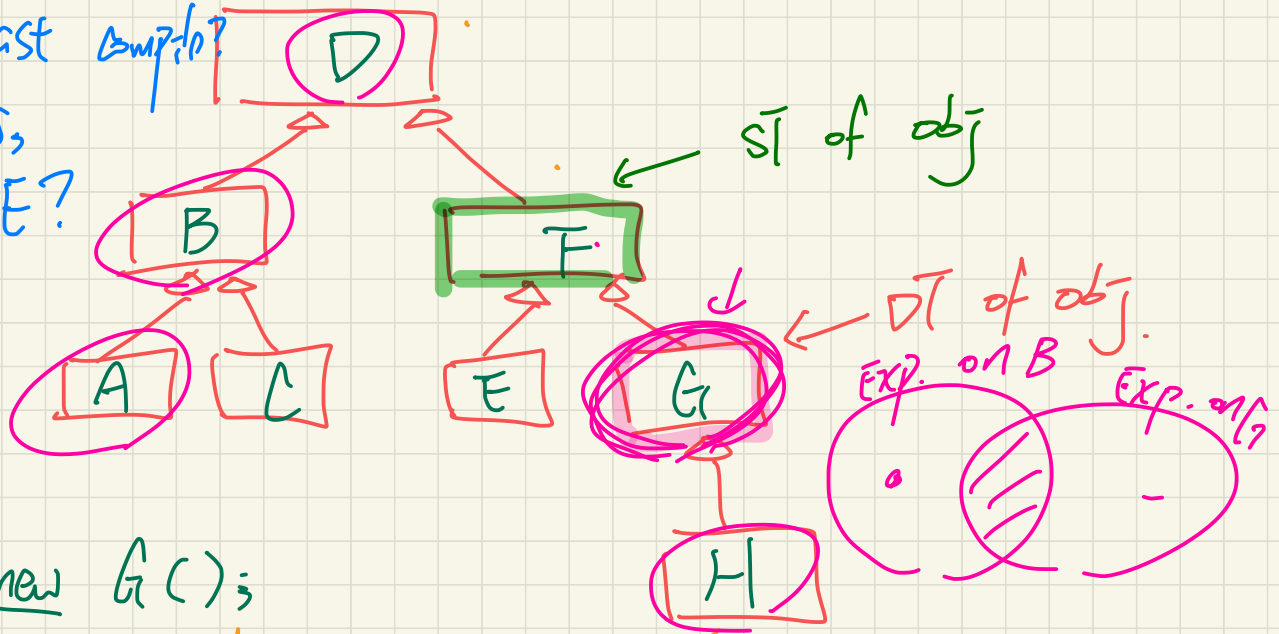


Bucket Array.

ArrayList<ArrayList<Entry>>

Q1. does the cast compile?

Q2. If it compiles, do we have CCE?




```
F obj = new G();
```

① (D) obj ✓ :: upward.

② (G) obj ✓ :: downward

③ (H) obj ✓ Compiles :: down — ✗ CCE occ.

④ (B) obj

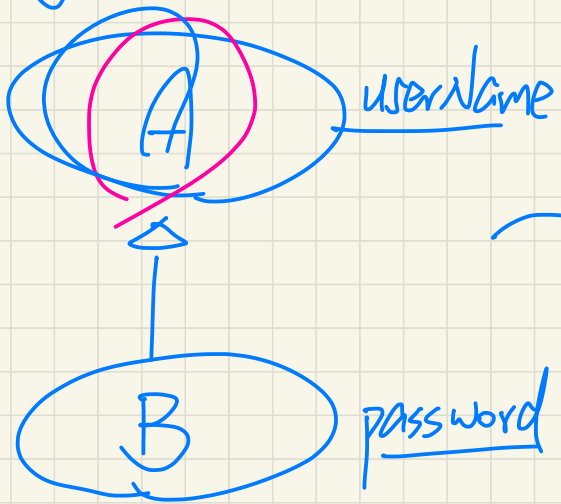
⑤ (A) obj

When do we get a CCE?

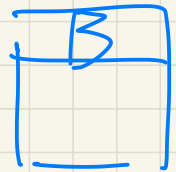
C (obj) dec. of C.

1. When DT of obj is not exp.
2. When DT of obj cannot fulfill C.

Upward Casting : Access Control



retrieveObj().un
• password



caller would only retrieve an object of static type A.

```
B obj = new B();
obj.un
obj.pw
}
```

```
A retrieveObj() {
(A) obj;
```

Call by Value: Primitive Argument

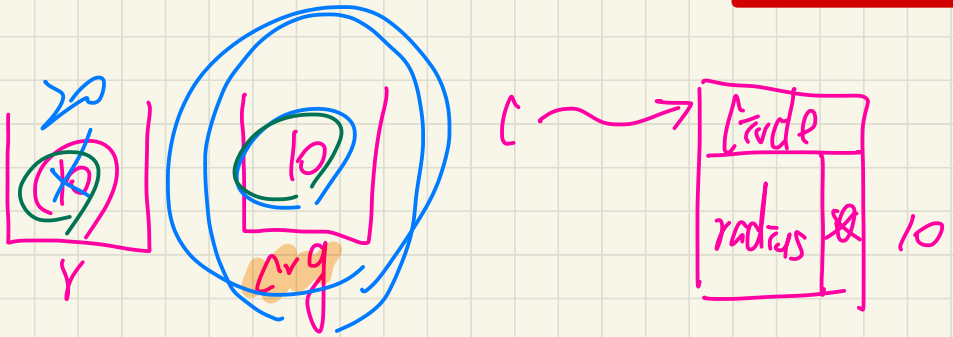
primitive parameter

```
class Circle {
   int radius;
   void setRadius(int r) {
      this.radius = r;
   }
}
```

r = arg

r = 20;

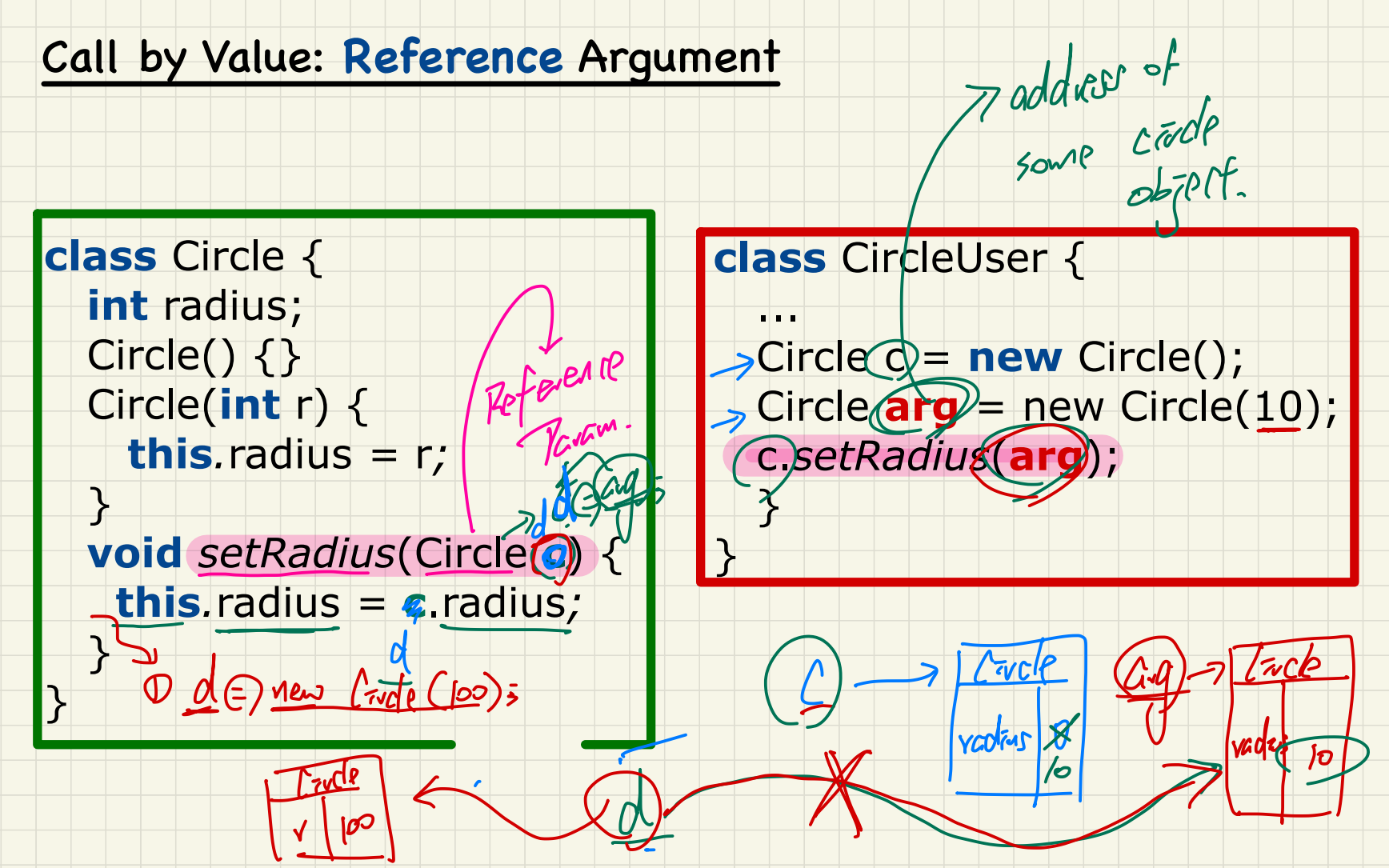
```
class CircleUser {
   ...
   Circle c = new Circle();
   int arg = 10;
   c.setRadius(arg);
   }
}
```



Call by Value: Reference Argument

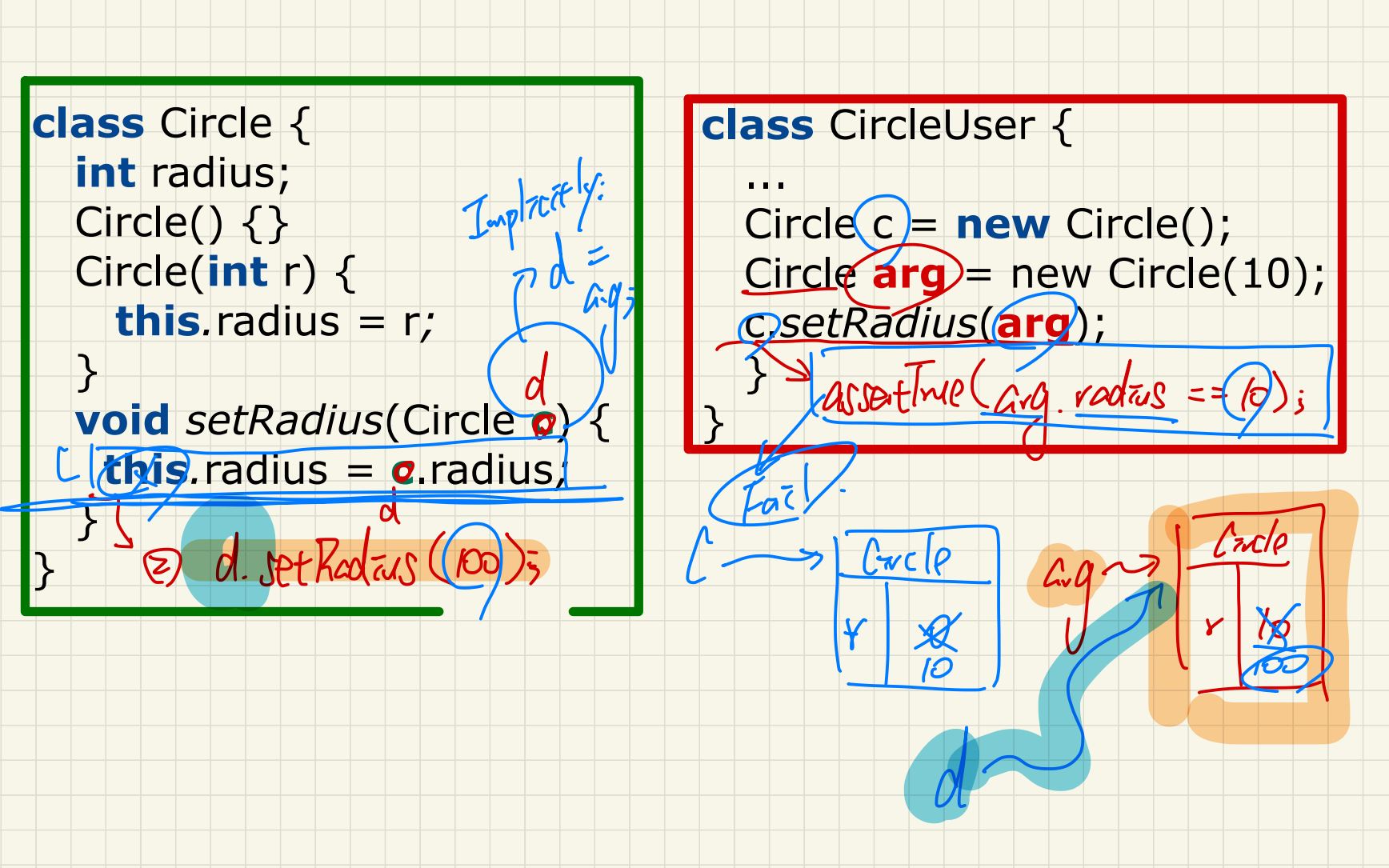
```
class Circle {
   int radius;
   Circle() {}
   Circle(int r) {
      this.radius = r;
   }
   void setRadius(Circle d) {
      this.radius = d.radius;
   }
}
```

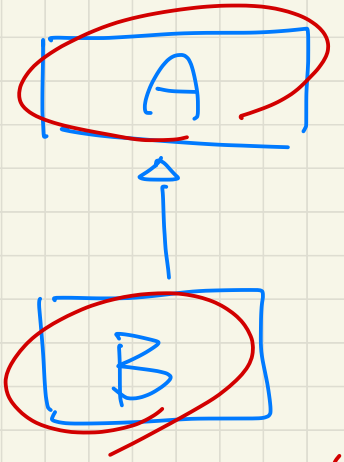
```
class CircleUser {
   ...
   Circle c = new Circle();
   Circle arg = new Circle(10);
   c.setRadius(arg);
   }
}
```



```
class Circle {
   int radius;
   Circle() {}
   Circle(int r) {
      this.radius = r;
   }
   void setRadius(Circle c) {
      this.radius = c.radius;
   }
}
```

```
class CircleUser {
   ...
   Circle c = new Circle();
   Circle arg = new Circle(10);
   c.setRadius(arg);
   }
}
```





① A oa = ...;

B ob = ...;

oa = ob;

✗ ob = oa;

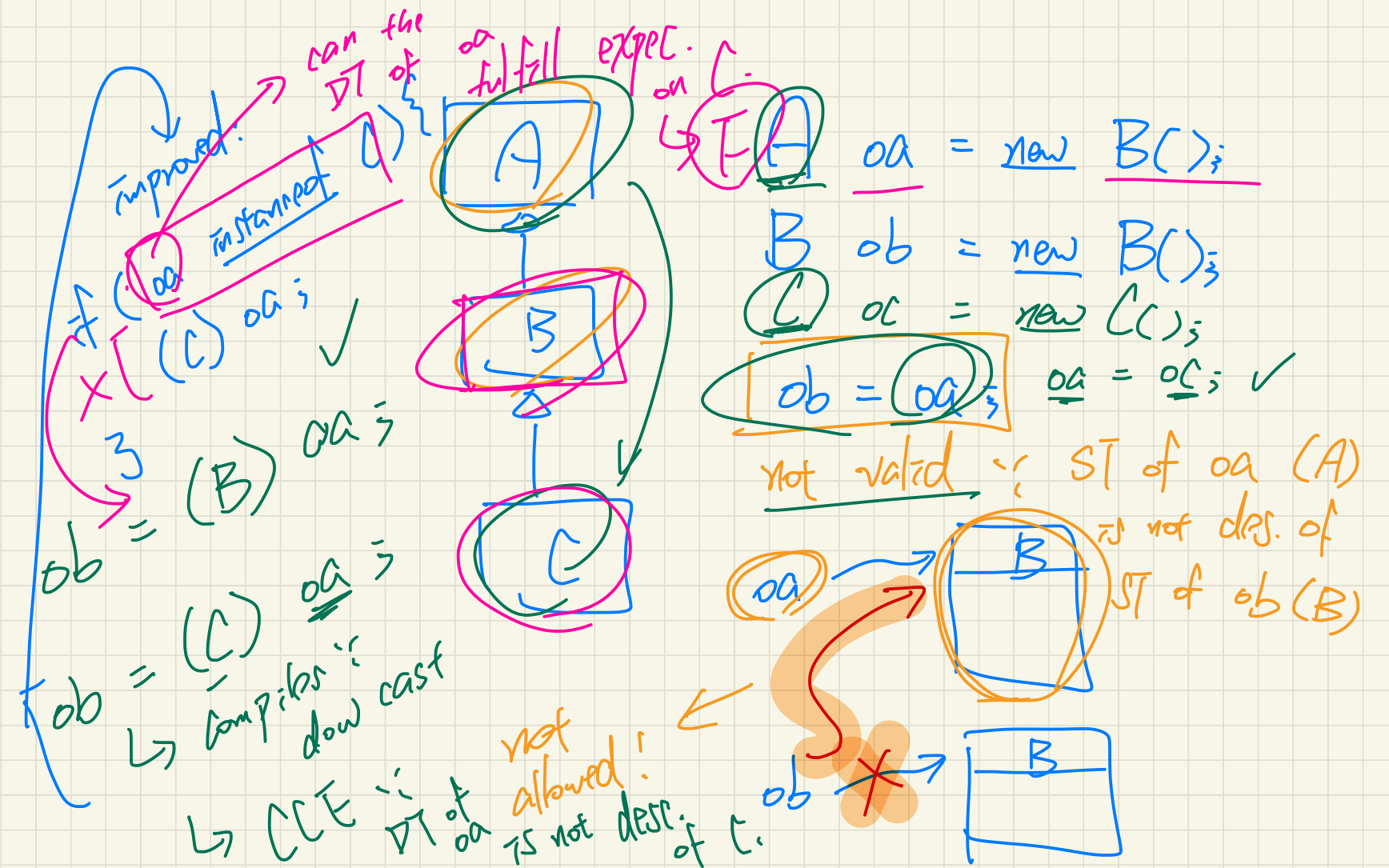
not compile

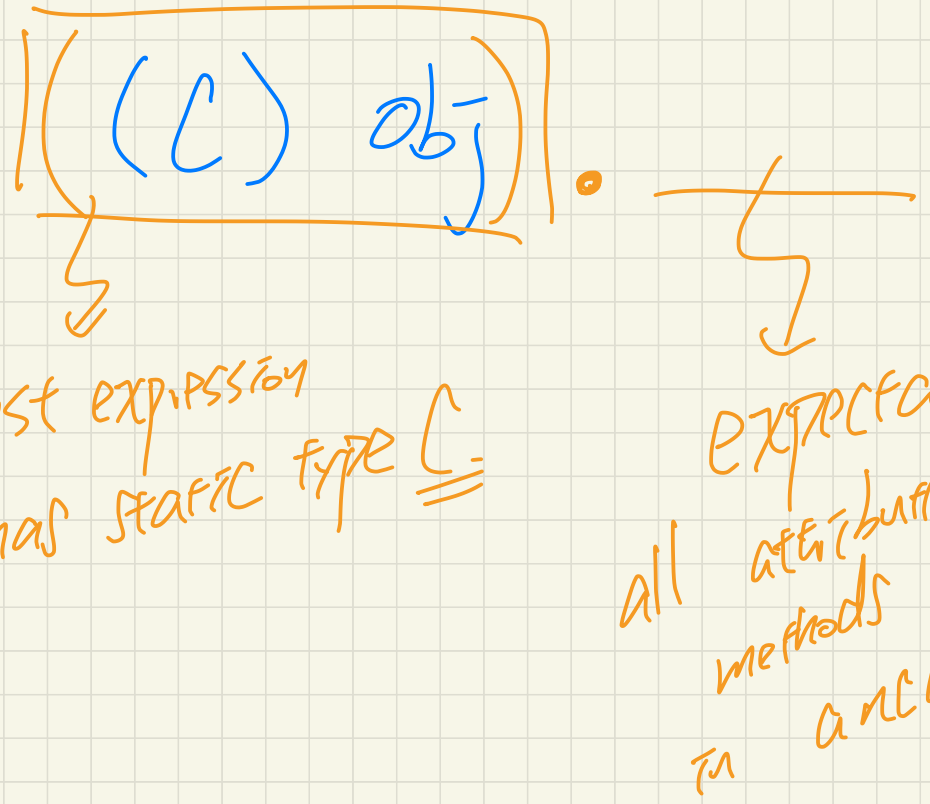
ST: B ST: A

② ✓ A oa = new B();

new object which can fulfill exp. on B.

B ob = new A();


improved:
can the DT of oa fulfill expec. on C.
instanceof
oa
if ( ) {
(C) oa;
X
3
(B) oa;
ob =
ob = (C) oa;
↳ compiles: down cast
↳ CCE: DT of oa is not desc. of C.
not allowed!
A
B
C
oa = new B();
B ob = new B();
C oc = new C();
ob = oa;
oa = oc; ✓
not valid: ST of oa (A) is not desc. of ST of ob (B)
oa
ob
B
B


(C) obj
cast expression
has static type C
expressions on C:
all attributes and
methods declared
in ancestors of C

Generic Parameters: ArrayList

```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

declare a type parameter to be used in ArrayList class

list1.add(new Point(3, 4)); X

String s = list2.remove(2); X

Caller of ArrayList

```
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<Point> list2 = new ArrayList<Point>();
```

```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

String String String String

```
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

Point Point Point Point