LECTURE 12

MONDAY OCTOBER 21

Method Call: Callee vs. Caller

```
class A {
  ...
  void m(T param) {
    /* use of param */
  }
}
```

```
class B {
  ...
  void n(...){
    A co = new A();
    co.m(arg);
  }
}
```

parameter

argument

int i = 10;
m(i);

1. Primitive int bool double char

2. Reference

copy value of arg.

Call by Value: Re-Assigning Primitive Parameter

```java
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

param

```java
1  @Test
2  public void testCallByVal() {
3    Util u = new Util();
4    int i = 10;
5    assertTrue(i == 10);
6    u.reassignInt(i);
7    assertTrue(i == 10);
8  }
```

C.O.

After L6, will i's value be incremented?

j = i; j → parameter, i → argument

Call by Value: Re-Assigning Reference Parameter

```java
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

```java
1  @Test
2  public void testCallByRef_1() {
3    Util u = new Util();
4    Point p = new Point(3, 4);
5    Point refOfPBefore = p;
6    u.reassignRef(p);
7    assertTrue(p==refOfPBefore);
8    assertTrue(p.x==3 && p.y==4);
9  }
```

```java
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

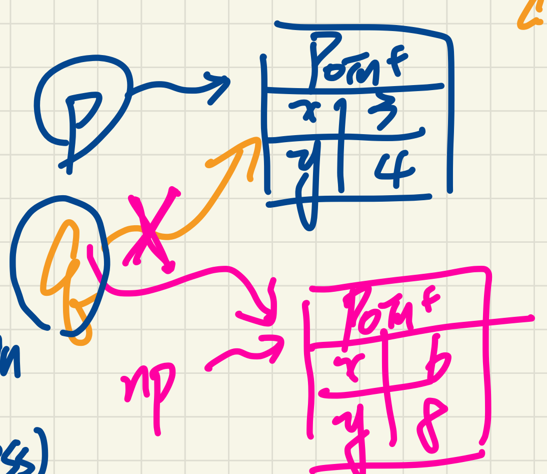
par.

arg

After L6, is p going to point to the same object?

q = p

When the param is of ref. type, do not try to re-assign it (∵ it's useless)



Call by Value: Calling Mutator on Reference Parameter

```java
public class Util {
  void reassignInt(int j) {
    j = j + 1; }
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np; }
  void changeViaRef(Point q) {
    q.moveHorizontally(3);
    q.moveVertically(4); } }
```

q = p

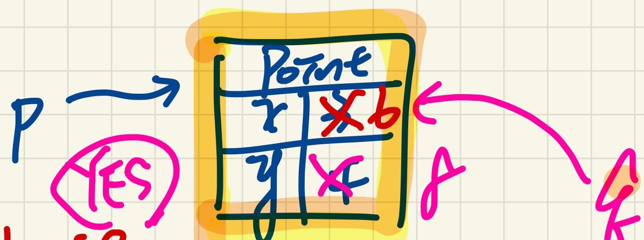
```java
1 @Test
2 public void testCallByRef_2() {
3   Util u = new Util();
4   Point p = new Point(3, 4);
5   Point refOfPBefore = p;
6   u.changeViaRef(p);
7   assertTrue(p==refOfPBefore);
8   assertTrue(p.x==6 && p.y==8);
9 }
```

```java
class Point {
  int x;
  int y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  void moveVertically(int y){
    this.y += y;
  }
  void moveHorizontally(int x){
    this.x += x;
  }
}
```

After L6:

(a) Is p pointing to the same object? YES

(b) Is the object pointed to by p initially modified? YES



API: ArrayList

int	`size()` Returns the number of elements in this list.
boolean	`add(E e)` Appends the specified element to the end of this list.
void	`add(int index, E element)` Inserts the specified element at the specified position in this list.
boolean	`contains(Object o)` Returns `true` if this list contains the specified element.
E	`remove(int index)` Removes the element at the specified position in this list.
boolean	`remove(Object o)` Removes the first occurrence of the specified element from this list, if it is present.
int	`indexOf(Object o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
E	`get(int index)` Returns the element at the specified position in this list.

true
false

used as param. types

used as return types

Generic Parameters: ArrayList

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

declaring a g.p. <E>

usages of E.

generic parameter for some type that will be instantiated by users of ArrayList.

Caller of ArrayList

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

user1

user2

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

ArrayList<Object> list3 = new ---;

① list1.add(new Point(3,4)); ✗

② list1.add("(3,4)");

③ list2.add(new Point(3,4));

④ list2.add("(3,4)"); ✗

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

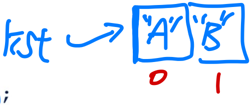
```java
class ArrayList<E> {
  boolean add(E e)
  E remove(int index)
  E get(int index)
}
```

E → String

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

E → Point

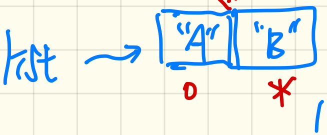
Use of ArrayList




```java
import java.util.ArrayList;
public class ArrayListTester {
  public static void main(String[] args) {
    ArrayList<String> list = new ArrayList<String>();
    println(list.size());
    println(list.contains("A"));
    println(list.indexOf("A"));
    list.add("A");
    list.add("B");
    println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
    println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
    list.add(1, "C");
    println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
    println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
    list.remove("C");
    println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
    println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));

    for(int i = 0; i < list.size(); i ++) {
      println(list.get(i));
    }
  }
}
```

list.length X



Hash Table

- 2-column table
- **keys** contain no duplicates
- **Values** may contain duplicates
- A **key** is used to identify a unique row

keys	values
"Alan"	"A"
"Mark"	"B+"
"Tom"	"A"
~~"Mark"~~	

API: HashTable

two generic param: K & V

int	size() Returns the number of keys in this hashtable.
boolean	containsKey(Object key) Tests if the specified object is a key in this hashtable.
boolean	containsValue(Object value) Returns true if this hashtable maps one or more keys to this value.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	put(K key, V value) Maps the specified key to the specified value in this hashtable.
V	remove(Object key) Removes the key (and its corresponding value) from this hashtable.

Generic Parameters: Hashtable

```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

generic parameters

< . - - >

usages of g.p.

Caller of Hashtable

```
Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```

```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

t1.get("alan") vs. t2.get(34)

```
Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```

```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

① t1.put("alan", 34); ✓

② t1.put(34, "alan"); ✗

③ t2.put("alan", 34); ✗

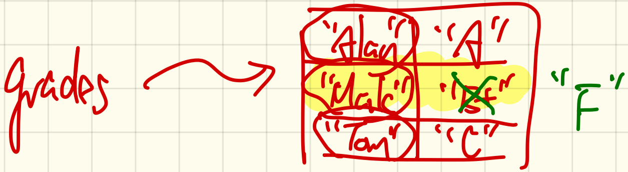
④ t2.put(34, "alan"); ✓

Use of HashTable

```java
import java.util.Hashtable;
public class HashTableTester {
  public static void main(String[] args) {
    Hashtable<String, String> grades = new Hashtable<String, String>();
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    grades.put("Alan", "A");
    grades.put("Mark", "B+");
    grades.put("Tom", "C");
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Key Mark exists: " + grades.containsKey("Mark"));
    System.out.println("Key Tom exists: " + grades.containsKey("Tom"));
    System.out.println("Key Simon exists: " + grades.containsKey("Simon"));
    System.out.println("Value A exists: " + grades.containsValue("A"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    System.out.println("Value C exists: " + grades.containsValue("C"));
    System.out.println("Value A+ exists: " + grades.containsValue("A+"));
    System.out.println("Value of existing key Alan: " + grades.get("Alan"));
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    System.out.println("Value of existing key Tom: " + grades.get("Tom"));
    System.out.println("Value of non-existing key Simon: " + grades.get("Simon"));
    grades.put("Mark", "F");
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    grades.remove("Alan");
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value of non-existing key Alan: " + grades.get("Alan"));
    }
```

empty → left column, right column

"Mark" is an existing key, overwrite its value.



```
class Hashtable<K, V> {
  V put(K key, V value)
  V get(Object key)
}
```

K

do this if you're defining your own.

gen. p. -

Solving a Problem Recursively

vs. iteratively - divide-and-conquer.

Given a **small** problem: | Solve it **directly**:

Given a **big** problem:

Divide it into (strictly) **smaller** problems:

Assume solutions to **smaller** problems:

Combine solutions to **smaller** problems:

```
m (i) {
  if (i == ...) { /* base case: do something directly */ }
  else {
    m (j);/* recursive call with strictly smaller value */
  }
}
```

recursive call to the same method

Fibonacci number.

1 1 2 3 5 . — —

- factorial n!
- Fibonacci number

→ ① loop implement?

② recursive methods ?