JAVA Programming

Type System

Overview

- Java Type System
- Primitive Types
- Boxing UnBoxing
- Common Operators
- CompoundAssignments
- Data Type Conversions
- Overflow
- Enum
- Reference Types
- Java Memory Model
- Stack and Heap
- Garbage Collector

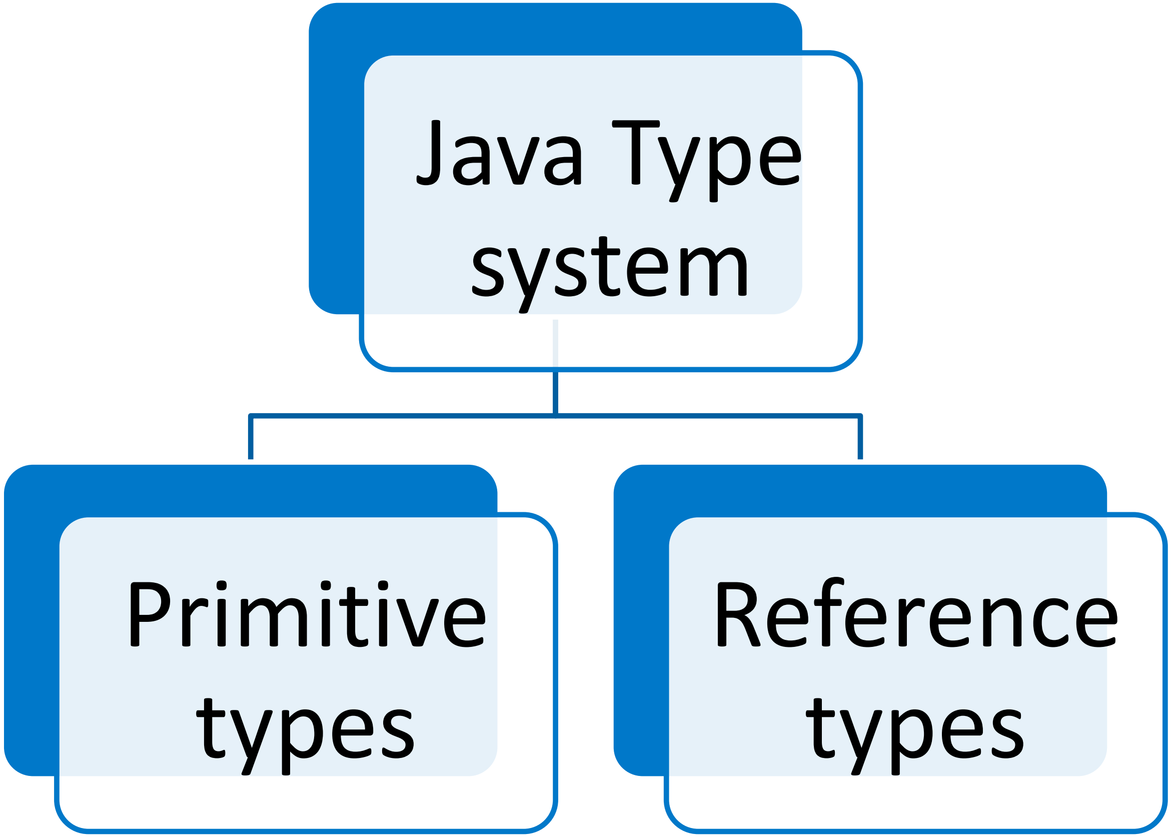
Java Type System

- Java uses Static Typing
  - Types are checked compile time

- The contrary is Dynamic Typing
  - Types are checked run time

Java Type System

- In Java we must specify the type of an object
- Predefined types



Java Type System

- Primitive Types
  - Directly contain data
  - Can not be null
- Reference Types
  - Contain a reference to an object
  - Can be null
  - Are Garbage Collected

Java Type System

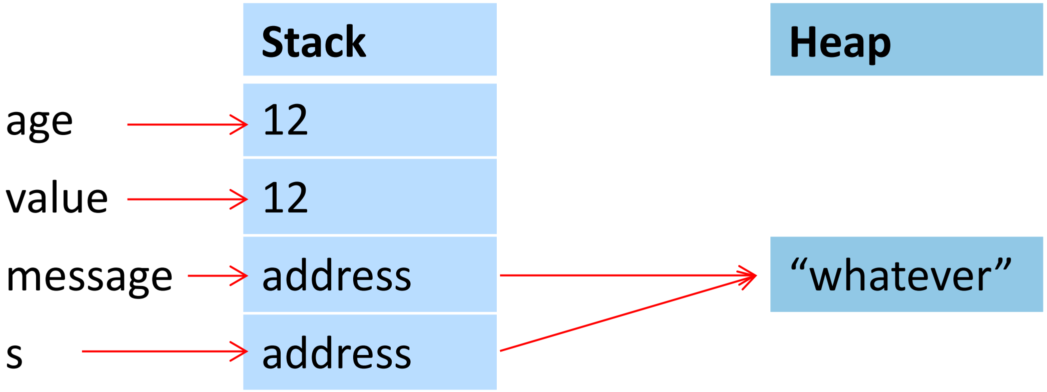
Primitive type

Reference type

```
private int age=12;
private int value=age;

private String message="whatever";

private String s=message;
```



Naming: Use descriptive names

Use camelCasing

No reserved key words

Primitive types

All Lowercase

1	byte	numeric	
2	short		
4	int		
8	long		
8	double		decimal
4	float		
	boolean	***true*** or ***false***	
2	char	***one char 'a'***	

Primitive types

```
int number = 1;
long number = 1;
```

```
long number = (long) 1;
```

```
long number = 1L;
```

```
double number = 1.0;
float number = 1.0; ✗
```

```
float number = (float) 1.0;
```

```
float number = 1.0f;
```

Primitive Types boolean

```
public void testBoolean() {
  boolean executeTest = false;
  boolean fiveLowerThenOne = (5 < 1);

  boolean executeTest; ✗
  executeTest = true;

  System.out.println(fiveLowerThenOne);
}
```

Primitive Types char

```
public void testChar() {
  char char1 = '1';
  char char2 = '2';

  System.out.println(char1 + char2);
}
```

+ operator

Output: 99 **(addition of ascii values)**

+ operator overloaded

```
 System.out.println(“” + char1 + char2);
```

Output: 12 **(concatenation)**

Boxing and Unboxing

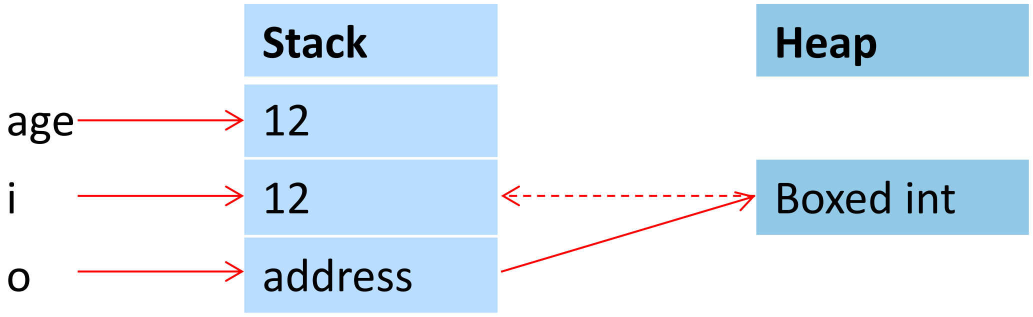
- Boxing
  - Allocates box, copies value into it
- Unboxing
  - Checks type of box, copies value out

```
private int age=12;
private Object o=age;
private int i=(Integer)o;
```

Boxing

UnBoxing

High performance impact!



Primitive-Wrapper types

- There are reasons to use objects in place of primitives
- Primitives are wrapped in an object
- Auto Boxing: a primitive is used where an object was expected.
- Auto UnBoxing: an object is used where a primitive was expected.

Primitive-Wrapper types

All Uppercase

- Byte
- Short
- Integer
- Long
- Double
- Float

Subclass of abstract **Number** class

- Boolean
- Character

Subclass of **Object** class

- All primitive-wrapper classes offer various methods (parse, convert, ….)

Primitive-Wrapper types

- Other subclasses of Number are
  - BigDecimal, BigInteger used for high-precision calculations
  - AtomicInteger, AtomicLong used in multi-threaded environments

Common Operators

Operators	Precedence
Postfix	expr++ expr--
Unuary	++expr --expr +expr -expr ~ !
Multiplicative	* / %
Additive	+ -
Shift	<< >> >>>
Relational	< > <= >= instanceof
equality	== !=
Bitwise AND	&
Bitwise exclusive OR	^
Bitwise inclusive OR	|
Logical AND	&&
Logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Compound assignments

- Four ways to increment by one

```
int age = 18;

age=age+1;
age+=1;
age++;
++age;
```

- Be ware of the differences

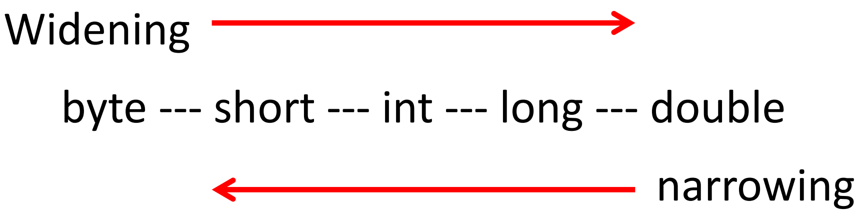
```
int age=18;
int incrementedAge= ++age;

age?
incrementedAge?
```

```
int age=18;
int incrementedAge= age++;

age?
incrementedAge?
```

Data Type Conversion



Widening

byte --- short --- int --- long --- double

narrowing

- Implicit (widening)

```
byte b=120;
int i=b;
```

- Explicit (narrowing)

```
long l=1234;
//int i=l;//cannot convert from long to int
int i=(int)l;//ok
```

Data Type Conversion

- Example: add two bytes

```
byte numberA = 1;
byte numberB = 0;
byte sum = numberA + numberB;
```

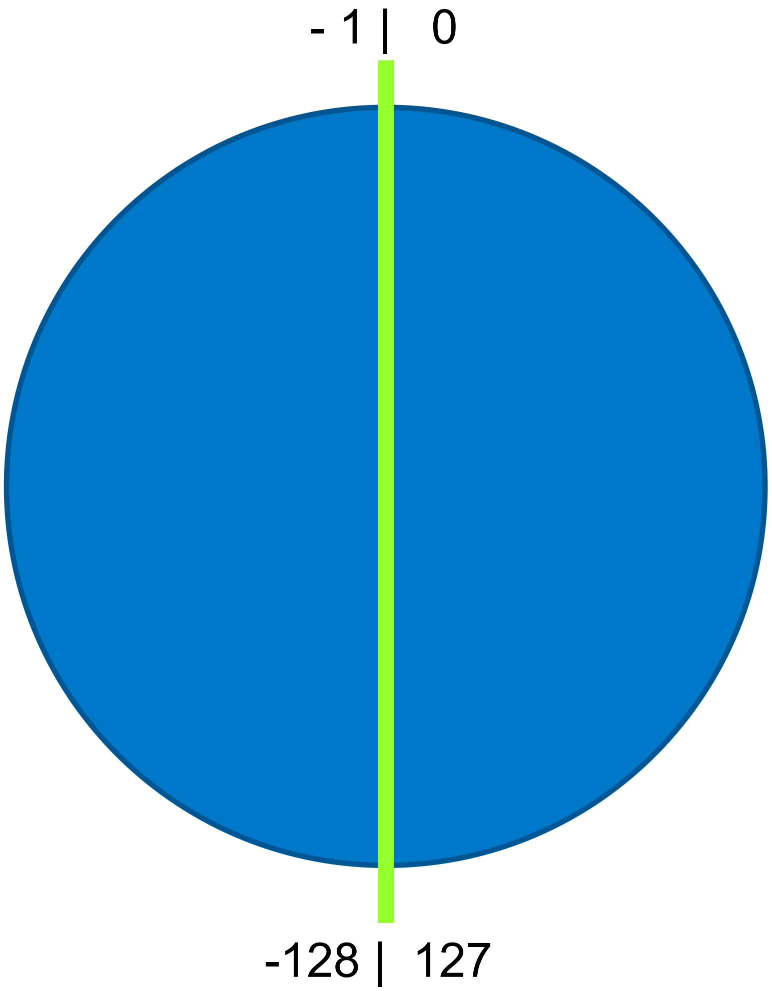
- Causes compile time error
- Implicit: integer type

```
byte numberA = 1;
byte numberB = 0;
byte sum = (byte) (numberA + numberB);
```

Cast

Overflow

Applies to byte, short, int and long



Two complements method

Sign bit

0	0	0	0	1	1
0	1	1	1	1	127
1	0	0	0	0	-128
1	1	1	1	1	-1

Overflow

```java
public void testOverflowOfBytes() {
  byte numberA = 127;
  byte numberB = 1;
  byte sum = (byte) (numberA + numberB);
  System.out.println("sum = " + sum);
}
```

Expected output?

Overflow

- Each primitive has a maximum value, because of the number of allocated bytes
- No warnings in case of overflow
- choose the most appropriate datatype

Enum

- Predefined set of constants
- Defining an enumeration type

uppercase

```
enum CompassPoint{NORTH,EAST,SOUTH,WEST};
```

- Using an enumeration type

```
CompassPoint compassPoint=CompassPoint.SOUTH;
System.out.println(compassPoint);//South
System.out.println(compassPoint.ordinal());//2 (North -> 0 )
```

Reference Types

- Defining a Reference Type

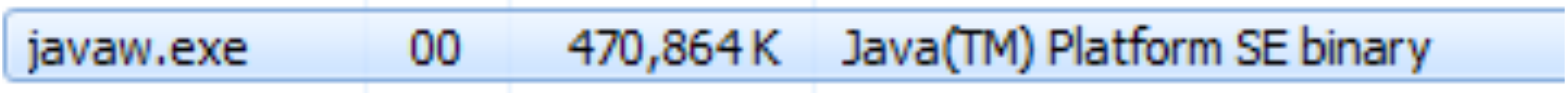
```
public class Employee{
    public String firstName;
    public int age;
}
```

- Using a Reference Type

```
Employee companyEmployee = new Employee();
companyEmployee.firstName = "Joe";
companyEmployee.age = 23;
```

Java memory model

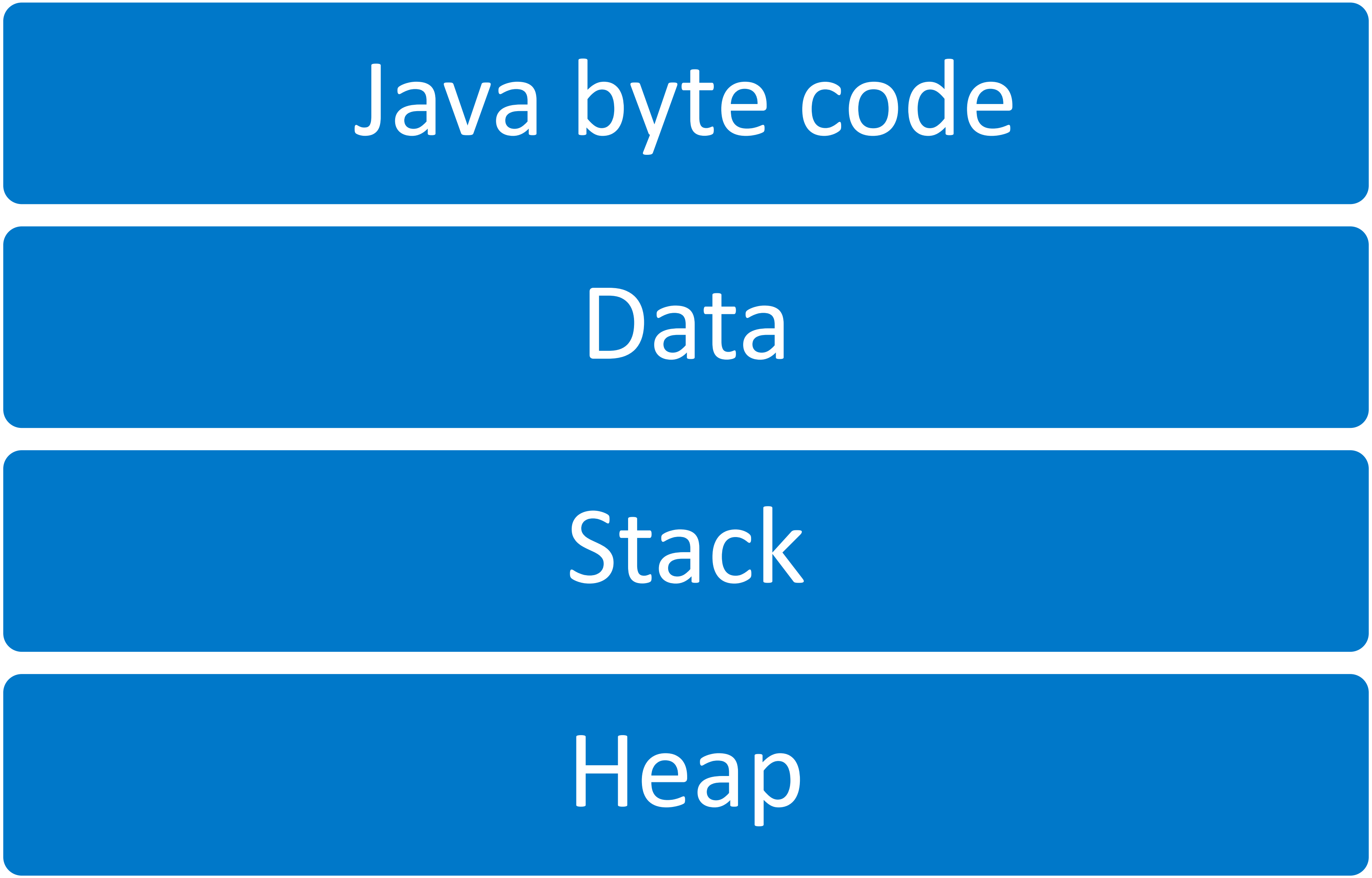
- Many classes are provided with a JRE



javaw.exe 00 470,864 K Java(TM) Platform SE binary

- Memory use: 470 mb
- How is it organized?

Java memory model



Java byte code

Data

Stack

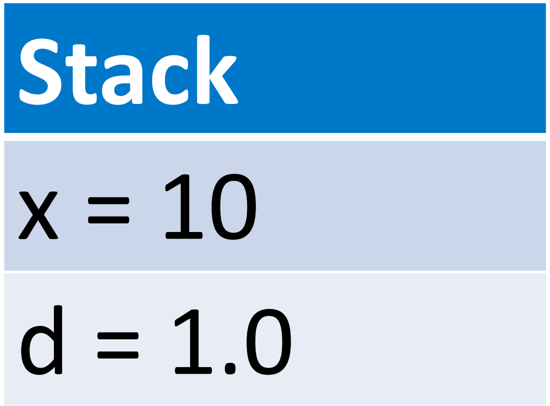
Heap

Stack and Heap

- Primitive types reside on the stack
- Parameters are passed to methods using the stack

```
func(10, 1.0);
```

```
void func(int x, double d) {
      … do something with
      … x and d
}
```



Stack

x = 10

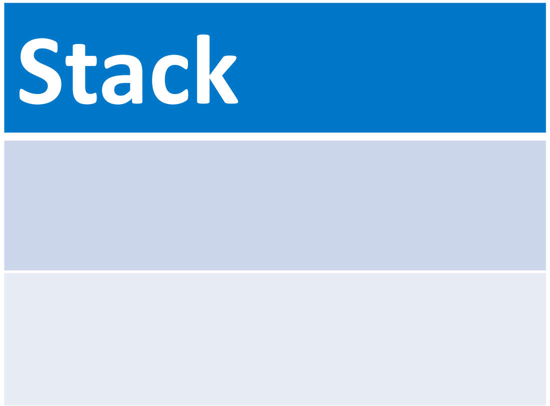
d = 1.0

Stack and Heap

- Primitive types reside on the stack
- Parameters are passed to methods using the stack

```
func(10, 1.0);
```

```
void func(int x, double d) {
        … do something with
        … x and d
}
```

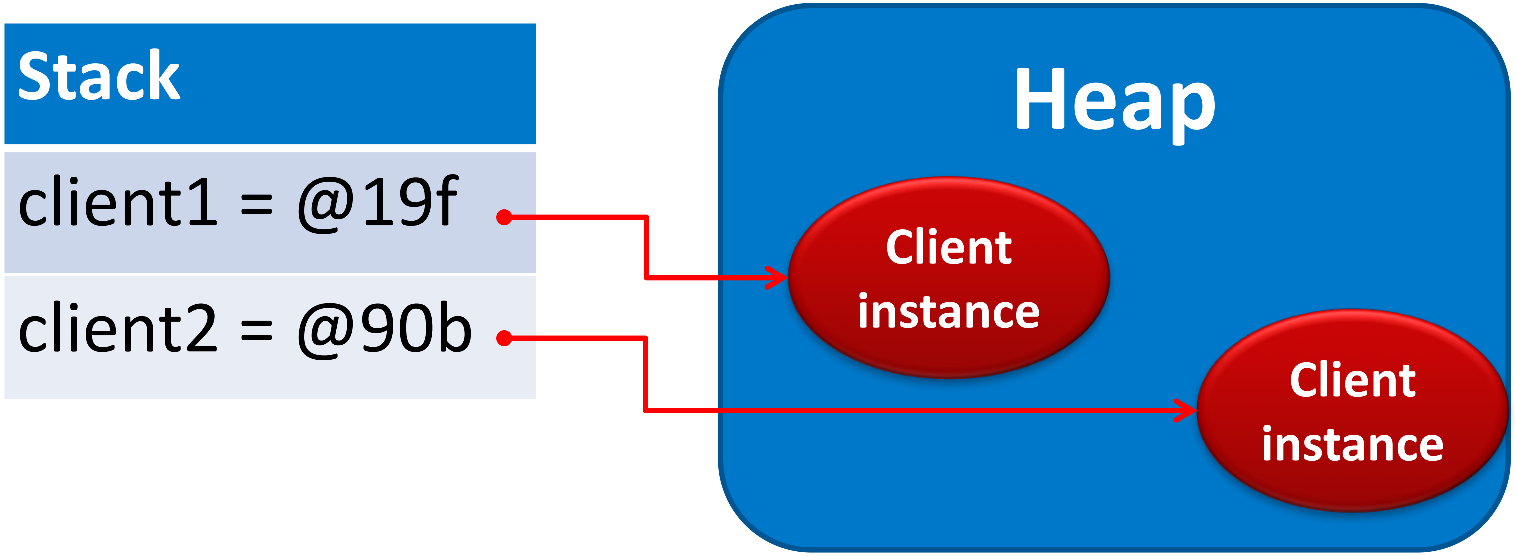


Stack

- When method ends, the parameters are removed from the stack.

Stack and Heap

- Reference types are stored on the heap
- Stack holds a reference to the reference type



Stack

client1 = @19f

client2 = @90b

Heap

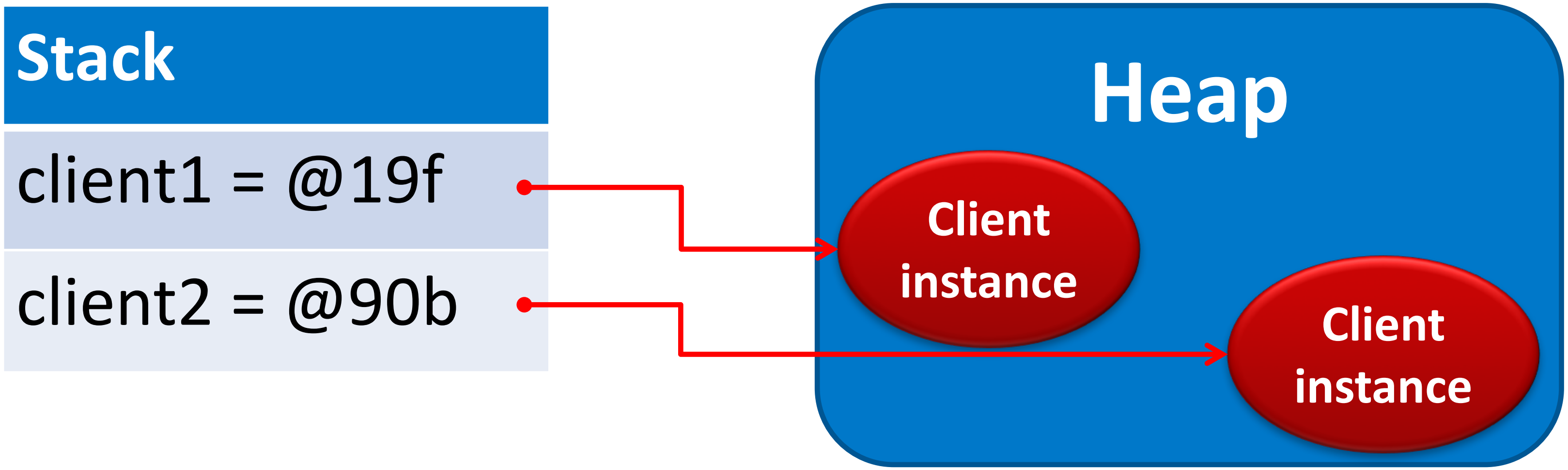
Client instance

Client instance

```
Client client1 = new Client(“Jan”);
Client client2 = new Client(“Piet”);
```

Stack and Heap

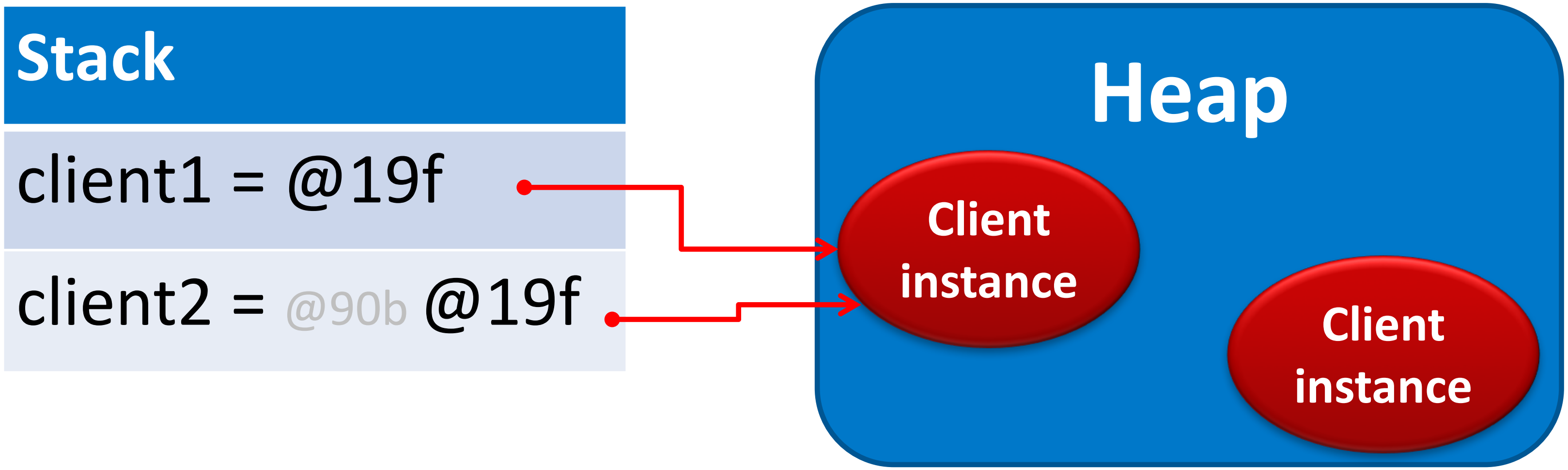
- Stack and heap



```
Client client1 = new Client(“Jan”);
Client client2 = new Client(“Piet”);
```

Stack and Heap

- Stack and heap



Stack

client1 = @19f

client2 = @90b @19f

Heap

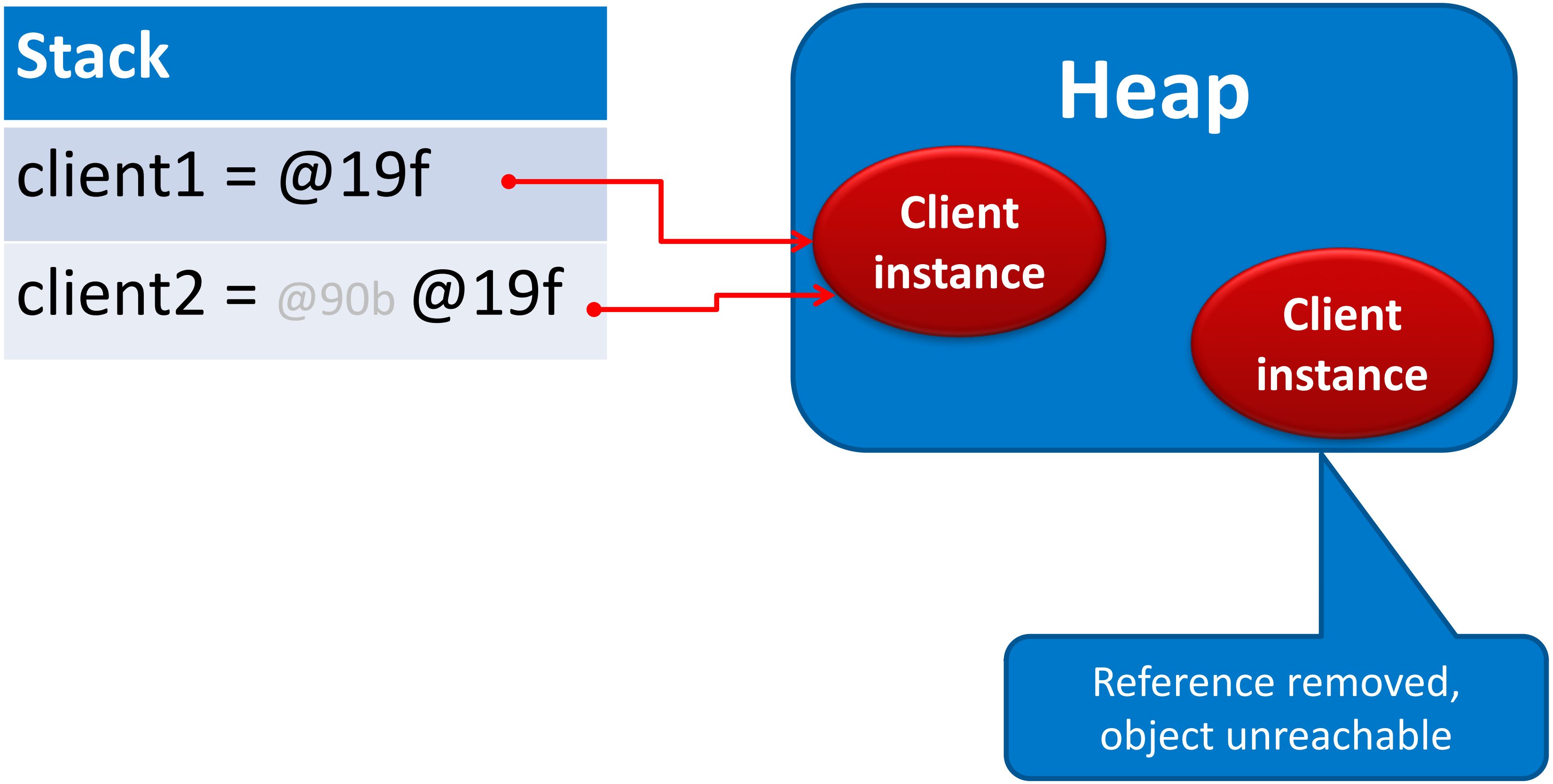
Client instance

Client instance

```
Client client1 = new Client(“Jan”);
Client client2 = client1;
```

Stack and Heap

- Stack and heap



Stack
client1 = @19f
client2 = @90b @19f

Reference removed, object unreachable

Garbage collector

- Java has a garbage collector
- The garbage collector “automatically” removes unreferenced objects

Lab

No lab associated with this module