Java SE 7

Module 1
Java Basics

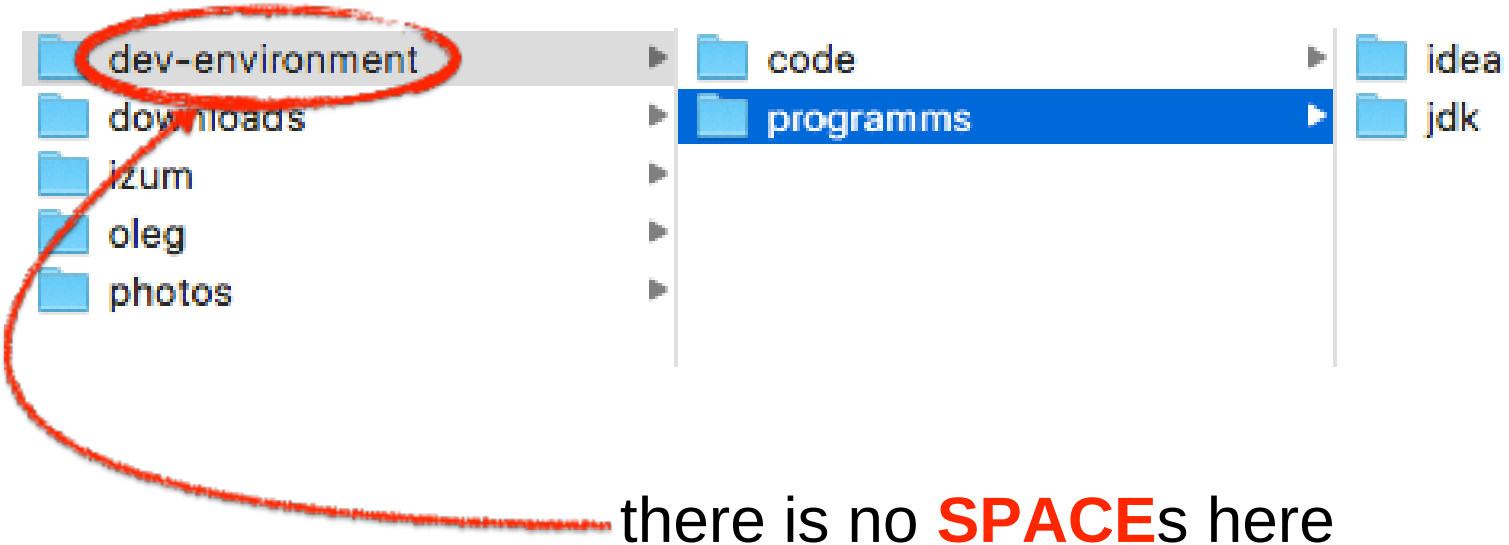
LUXOFT TRAINING

LXFT LISTED NYSE

Tools

- JDK (Java Development Kit)
- IntelliJ IDEA (The best tool for Java coding)

Folder Structure



IntelliJ IDEA Usage

- Create new project
- Syntax highlighting
- Suggestions
- How to run your code

```
public class FileStructure
{

    public static void main(String[] args)
    {
        // comments looks like this

        // your code here between { }
    }

}
```

```
public class FileStructure
{

    public static void main(String[] args)
    {
        // comments looks like this

        // your code here between { }
    }

}
```

```
public class FileStructure
{

    public static void main(String[] args)
    {
        // comments looks like this

        // your code here between { }
    }

}
```

```
public class FileStructure
{

    public static void main(String[] args)
    {
        // comments looks like this

        // your code here between { }
    }

}
```

Numbers & Strings

Integer Math


```
public static void main(String[] args)
{
    System.out.println( 232 );
    System.out.println( 34 );
    System.out.println( 10 + 6 );
}
```

Quick Task

Calculate number of seconds in 9 weeks.

Integer Division


```
System.out.println( 10 / 3 );
```

Reminder

```
System.out.println( 10 % 4 );
```

Integer


```
public static void main(String[] args)
{
    System.out.println(Integer.MIN_VALUE);
    System.out.println(Integer.MAX_VALUE);
}
```

Integer


```
public static void main(String[] args)
{
    System.out.println(Integer.MIN_VALUE - 1);
}
```

Integer Math


```
public static void main(String[] args)
{
    System.out.println("Max + 1: " + (Integer.MAX_VALUE + 1));
}
```

Variable

[int] [int]

type nameCamelCase **=** <expression> **;**

Variable


```
int    countOfSeconds = 90;

int countOfHoursInOneWeek = 24 * 7;

int randomExpressionExample =
    9 * 34 - 14 * (43 + 90) / 80 % 3;
```

```
type nameCamelCase = <expression>;
```

Variables in memory

Type	Name	Value
int	countOfSeconds	60
int	countOfHoursInOneWeek	168
int	randomExpressionExample	304

Variable


```
int secondsInMinute = 60;

int tenMinutes = 10 * secondsInMinute;
```

```
type nameCamelCase = <expression>;
```

Quick Task

Calculate number of seconds in 9 weeks.

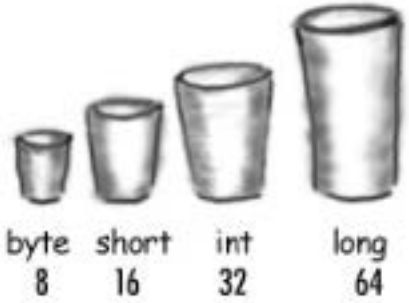
Please use variables now!

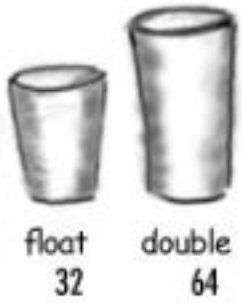
Real numbers


```
double randomPrice = 459.98;

double youCanAlsoDoTheMathHere = 34.9 + 98 * 6;

float iNeedFloat = 4.05f;
```

```
type nameCamelCase = <expression>;
```

Primitive types





boolean and char

boolean (JVM-specific) ***true*** or ***false***

char 16 bits 0 to 65535

numeric (all are signed)

integer

byte 8 bits -128 to 127

short 16 bits -32768 to 32767

int 32 bits -2147483648 to 2147483647

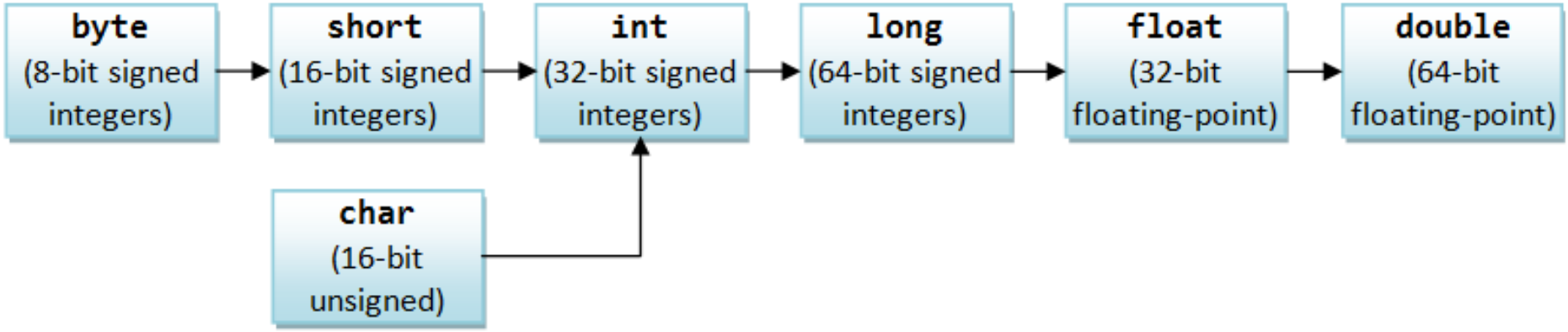
long 64 bits -huge to huge

floating point

float 32 bits varies

double 64 bits varies

Primitive data types casting



Orders of Implicit Type-Casting for Primitives

Primitive data types casting


```
int i = 19;
long l = 455L;

l = i;
i = (int) l;
```

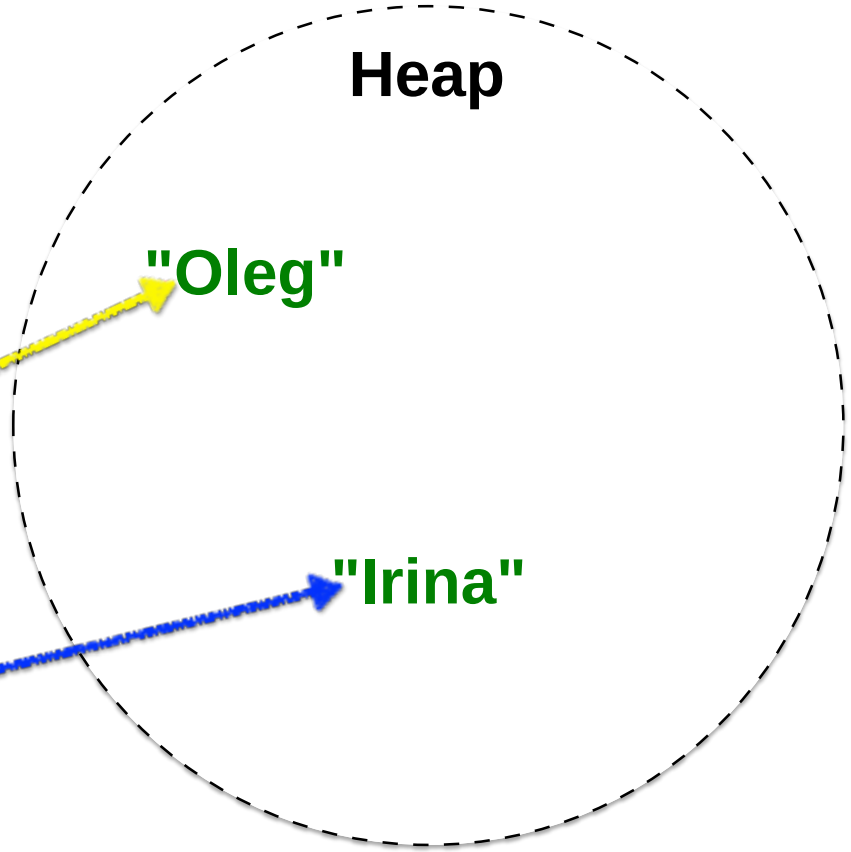
String

```
String oleg = new String("Oleg");

String irina = "Irina";
```

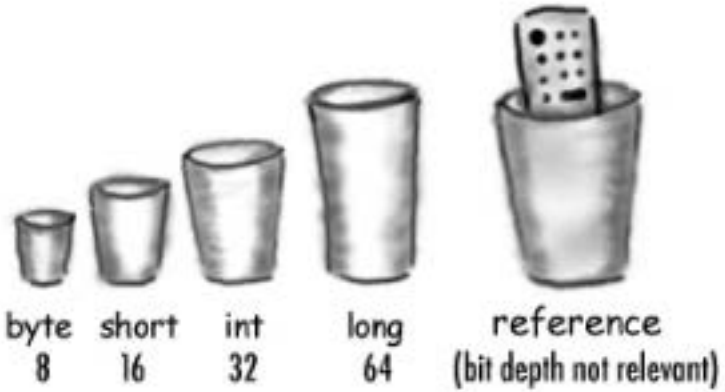
```
type nameCamelCase = <expression>;
```

String in memory

Type	Name	Value
String	oleg	Ref
String	irina	Ref



Reference Type



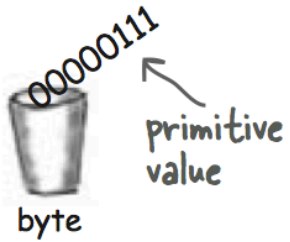
An object reference is just another variable value.

Something that goes in a cup.
Only this time, the value is a remote control.

Primitive Variable

```
byte x = 7;
```

The bits representing 7 go into the variable. (00000111).

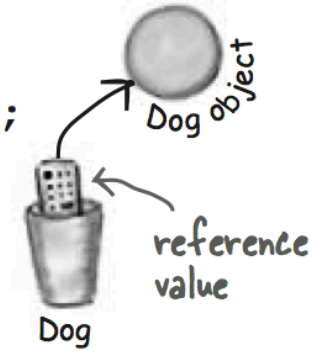


Reference Variable

```
Dog myDog = new Dog();
```

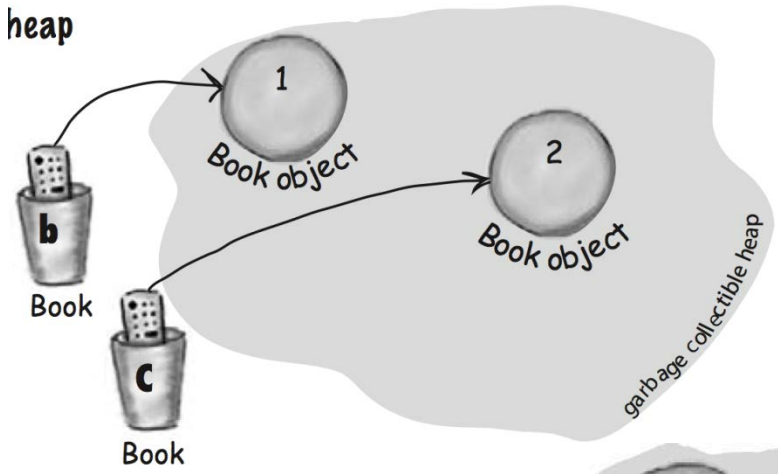
The bits representing a way to get to the Dog object go into the variable.

The Dog object itself does not go into the variable!



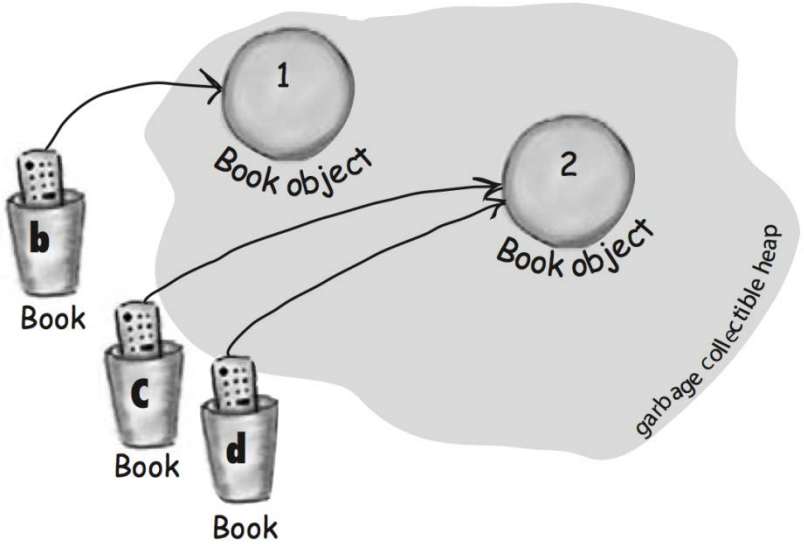
Heap life

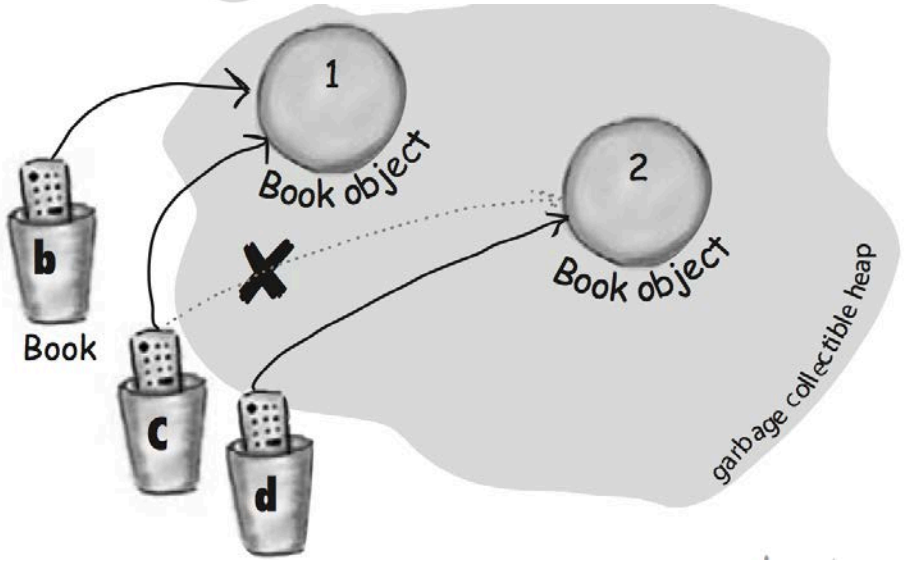
Book b = **new** Book();
Book c = **new** Book();



c = b;

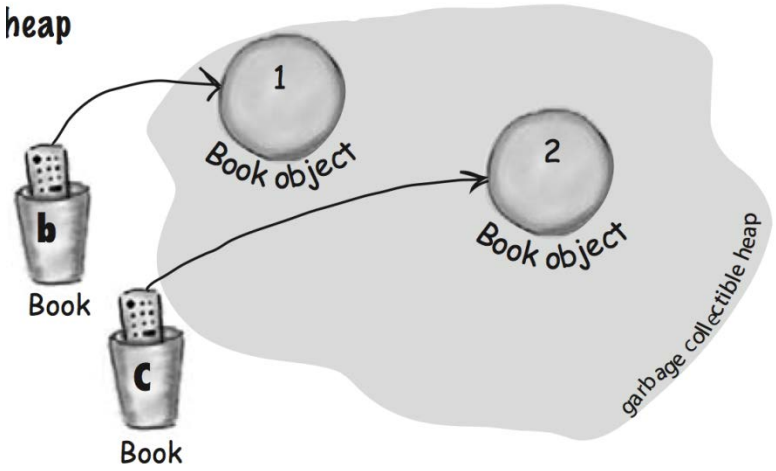
Book d = c;



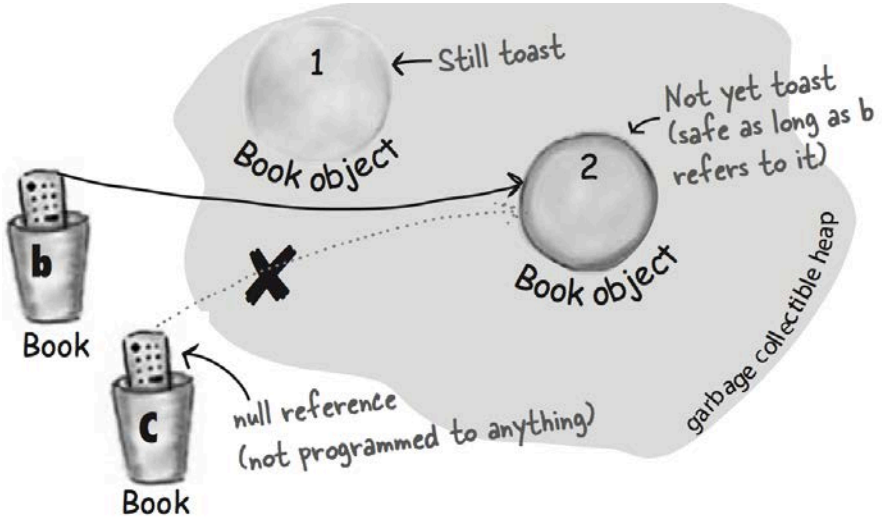


Heap life

Book b = **new** Book();
Book c = **new** Book();



b = c;
c = **null**;



String concatenation


```
String oleg = new String("Oleg");

System.out.println("0: " + oleg);

oleg += "!";

System.out.println("1: " + oleg);
```

Output:

0: Oleg

1: Oleg!

String length


```
String name = "Alexandra";

int length = name.length();

System.out.println(name + " contains "
        + length + " letters.");
```

Runtime exception


```
String str = "abcdefg...";

System.out.println("0: " + str);

str = null;
System.out.println("1: " + str.length());
```

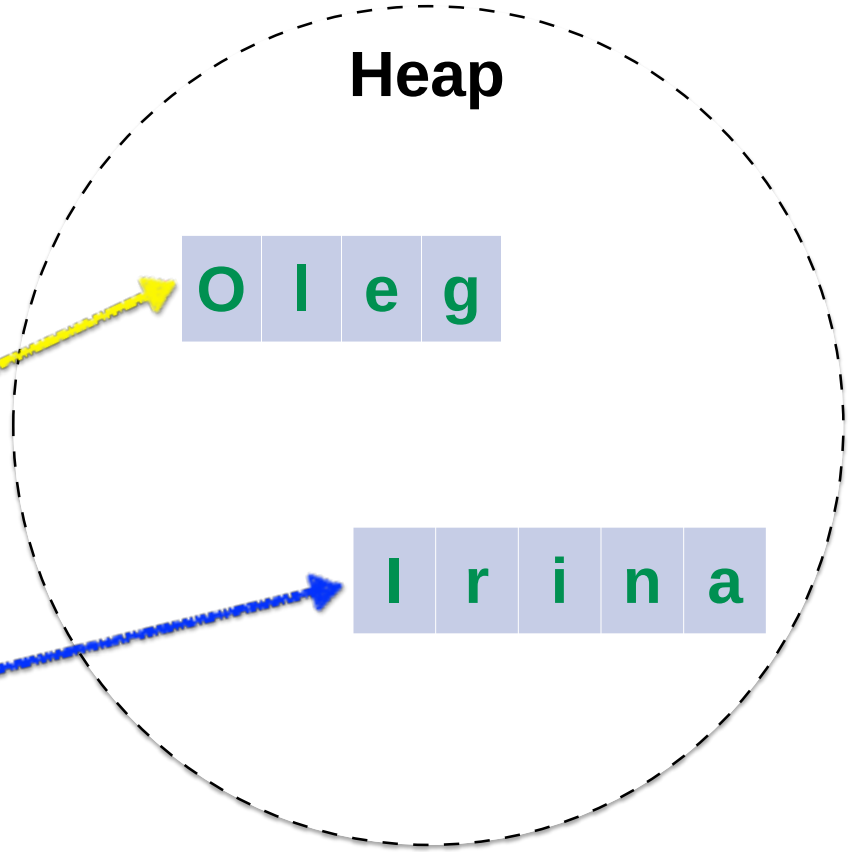
Type char


```
char a = 'a';
System.out.println(a);

char A = 65;
System.out.println(A);
```

Char range: 0 - 65535

String in memory

Type	Name	Value
String	oleg	Ref
String	irina	Ref



Chars in String


```
String galaxy = "Milky Way";

char ch = galaxy.charAt(0);
```

Chars in String


```
String galaxy = "Milky Way";

galaxy.indexOf('y');
```

int ch

```
galaxy.indexOf('y', 5);
```

int ch, **int** fromIndex

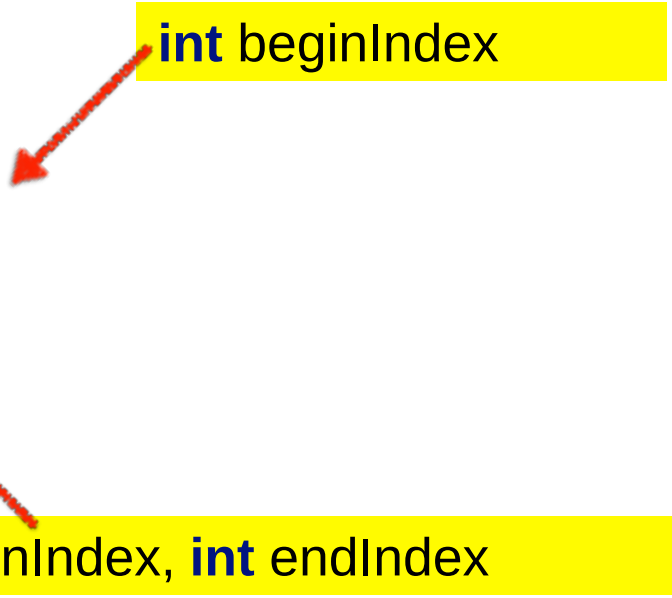
String comparison

Use **==** to check that variables referencing the same object.

Use **.equals** to check that objects equals accoring to our comparison rules.

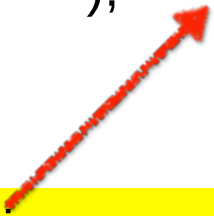
String methods, substring


```
String str = "Programming is cool.";

String substring = str.substring(3);

substring = str.substring(11, 14);
```



String methods, replace


```
String str = "Programming is cool.";

String replaced = str.replaceAll("is", "-");
```



String regex, String replacement

Quick Task

You have random **double** number. Write a program that prints to the output whole and fractional parts separately.

For example:

343.36	whole: 343, fract: 36
3.069	whole: 3, fract: 069
.2	whole: 0, fract: 2

Quick Task


```
  | A B C D E F G H I
--|-----------------
1 | B B B B B B B B B
2 |
3 | B B   B B B   B B
4 | B B   B B B   B B
5 | B B   B B B   B B
6 | B B B   B B     B
7 | B B B   B B     B
8 | B B B   B B     B
9 | B B B T E B B   B
```

B - Brick
T - Tank
E - Eagle (headquarters)

Methods

What method is?

Method signature

any existing java type or **void** *

any existing java type

```
public static return_type nameCamelCase(p1_type p1, p2_type p2)
{
    [return 0];
}
```

required if method **return_type** is not **void**

* **void** - means nothing to return.

Quick Task

Implement method **moveForward()**. Tank should move one quadrant right after each call.

Quick Task

Implement method **getTankQuadrant(int x, int y)**.

Method should return name of the quadrant according to given coordinates.

Method params.

How Java calls methods and send parameters to it.

Method overloaing.

You can declare methods with ientical name if param types are different.

```
public static int sum(int a, int b)
{
    return a + b;
}

public static long sum(long a, long b)
{
    return a + b;
}
```

Method overloaing.

You can declare methods with ientical name if param types are different.

```
public static int sum(int a, int b)
{
    return a + b;
}

public static int sum(int a, int b, int c)
{
    return a + b + c;
}
```

Visibility

Variable are visible and can be used within a block where it's declared.

if statement

any **boolean expression**

```
if (a % 2 == 0)
{
    a *= 2;
}
```

this code will be executed **only if** **boolean expression** returns **true**

if-else statement


```
if (a % 2 == 0)
{
        boolean expression == true
    a *= 2;
}
else
{
        boolean expression == false
    a /= 3;
}
```

else-if statement


```
if (a > 8)
{
    System.out.println("--> if a > 8");
    a += 3;
}
else if (a > 3)
{
    System.out.println("--> if a > 3");
    a += 2;
}
else
{
    System.out.println("--> else");
    a += 1;
}
```

while Loop


```
int counter = 0;

while (counter < 10)
{
    System.out.println(counter++);
}
```

any **boolean expression**

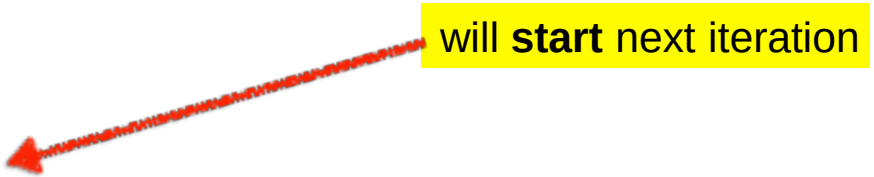
this code will be executed **until** **boolean expression** returns **true**

break


```
while (counter < totalCycles)
{
    if (counter == 7)
    {
        break;
    }
    System.out.println(counter++);
}
```

continue


```
while (counter < totalCycles)
{
    if (counter % 3 == 0)
    {
        counter++;
        continue;
    }

    System.out.println(counter++);
}
```



Quick Task

Implement logic that forces a tank moving forward and then backward forever.

Tank should slowly move from quadrant **A1** to **A9**, then from **A9** to **A1** and so on…

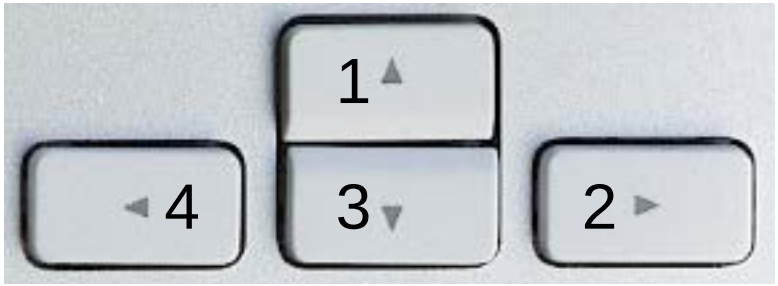
Quick Task

Implement method **void move(int direction)**.

One method call should smoothly move the tank one quarant according to given **direction** parameter value.

direction

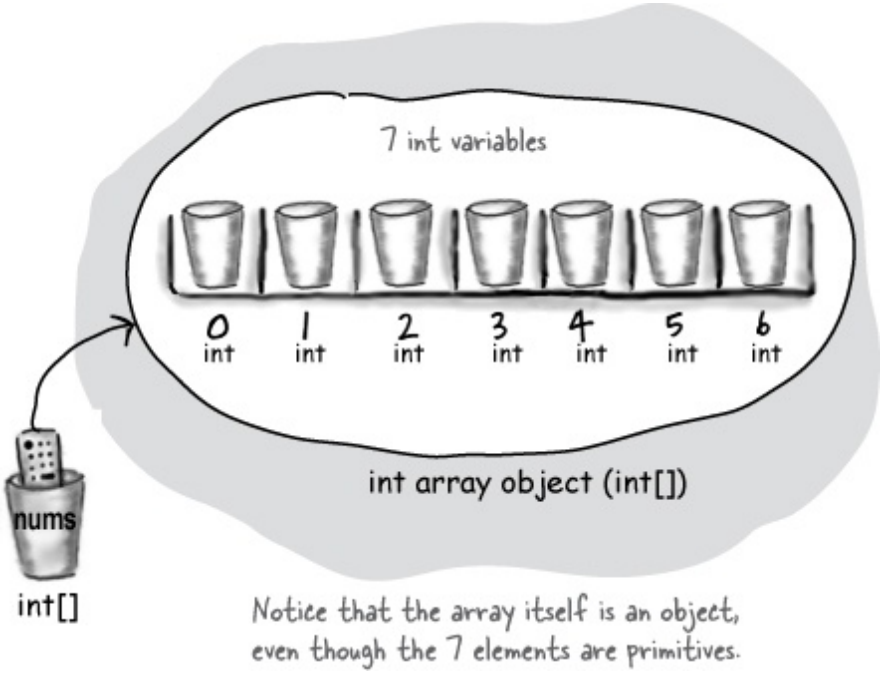
1 - up
2 - right
3 - down
4 - left



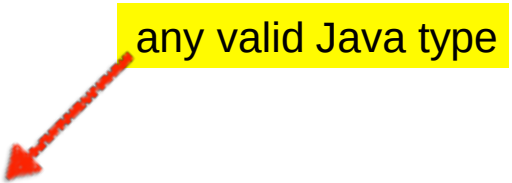
Arrays

What is array?

Data structure that keeps limited number of objects with defined type.



How to create array




```
type[] anyName = new type[number_of_objects];
```

How to create array, examples


```
int[] numbers = new int[10];

double[] prices = new double[10];

int[] ints = new int[] {1, 2, 3};

int[] data = {1, 2, 3, 4, 5, 6, 7};
```

Array, data access


```
int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

int num;

num = numbers[3];
num = numbers[7];

numbers[0] = 100;
numbers[6] = 600;
```

Array, data access

1. Declare an int array variable. An array variable is a remote control to an array object.

```
int[] nums;
```

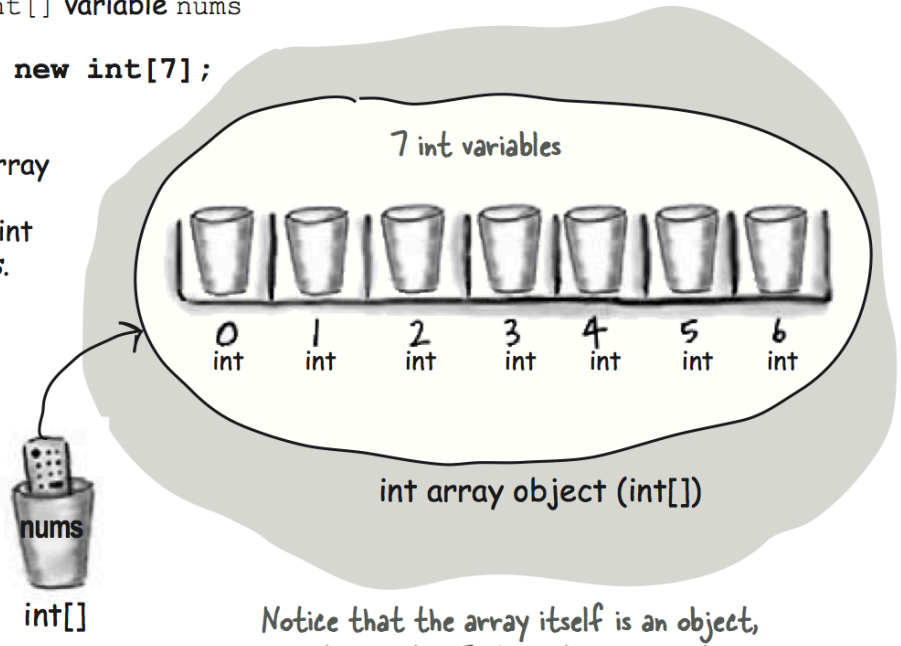
2. Create a new int array with a length of 7, and assign it to the previously-declared `int[]` variable `nums`

```
nums = new int[7];
```

3. Give each element in the array an int value.
Remember, elements in an int *array* are just int *variables*.

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```

7 int variables



Notice that the array itself is an object, even though the 7 elements are primitives.

What this code prints?


```
public static void main(String[] args)
{
    int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    setValue(numbers, 3, 300);
    System.out.println(Arrays.toString(numbers));
}

public static void setValue(int[] data, int idx, int value)
{
    data[idx] = value;
}
```

Quick Task

Implement method **void swap(int[] data, int idx1, int idx2)**.

Method should exchange values with **idx1** and **idx2** in **data** array.

for loop


```
int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

for (int i = 0; i < numbers.length; i++)
{
    int n = numbers[i];
    System.out.println(n);
}
```

- initialization
- condition
- increment

Quick Task, use for Loop


```
  | A B C D E F G H I
--|-------------------
1 | B B B B B B B B B
2 |
3 | B B   B B B   B B
4 | B B   B B B   B B
5 | B B   B B B   B B
6 | B B B   B B     B
7 | B B B   B B     B
8 | B B B   B B     B
9 | B B B T E B B   B
```

B - Brick
T - Tank
E - Eagle (headquarters)

for-each loop


```
int[] numbers = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

for (int n : numbers)
{
    System.out.println(n);
}
```

- data structure to iterate
- current element

Quick Task

Implement method **void sortAsc(int[] data)**.

Method should sort **data** array ascending order.

Array of Strings (objects)


```
String[] names = new String[10];
System.out.println(Arrays.toString(names));

names[3] = "Anna";
names[9] = "Max";
```

Array of Strings (objects)

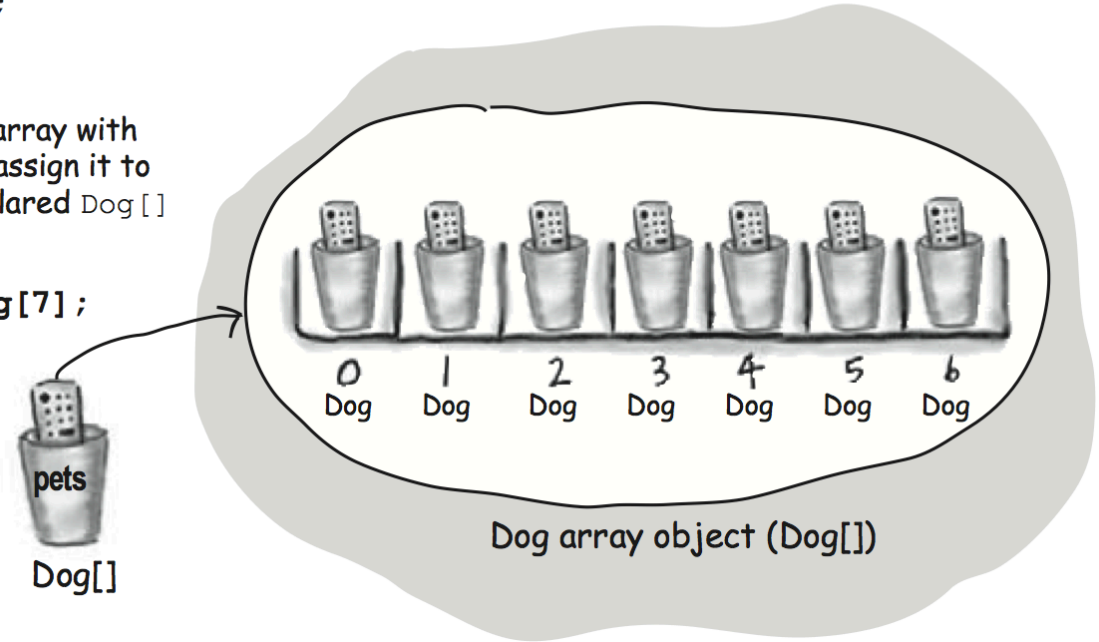
Make an array of Dogs

1. Declare a Dog array variable

```
Dog[] pets;
```

2. Create a new Dog array with a length of 7, and assign it to the previously-declared `Dog[]` variable `pets`

```
pets = new Dog[7];
```



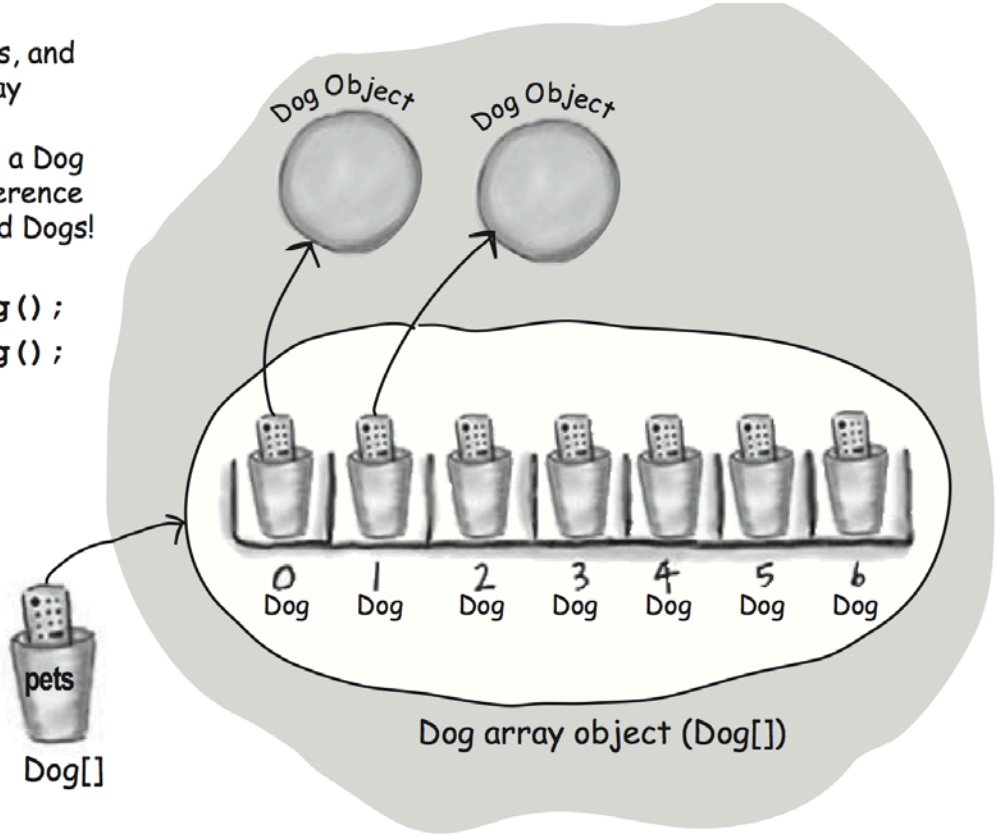
What's missing?

Dogs! We have an array of Dog *references*, but no actual Dog *objects*!

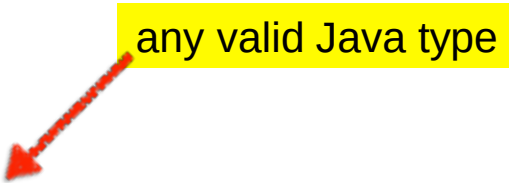
Array of Strings (objects)

3 Create new Dog objects, and assign them to the array elements.
Remember, elements in a Dog *array* are just Dog reference *variables*. We still need Dogs!

```
pets[0] = new Dog();
pets[1] = new Dog();
```



Multidimensional array




```
type[][] name = new type[number_of_objects][];
```

Multidimensional array, usage example


```
int[][] data = new int[10][];

data[0] = new int[] {1, 2, 3};
data[1] = new int[] {1, 2};

int[] numbers = data[0];

int number = data[0][1];

data[1][0] = 1000;
```

Quick Task, use String [][]


```
  | A B C D E F G H I
--|-----------------
1 | B B B B B B B B B
2 |
3 | B B   B B B   B B
4 | B B   B B B   B B
5 | B B   B B B   B B
6 | B B B   B B     B
7 | B B B   B B     B
8 | B B B   B B     B
9 | B B B T E B B   B
```

B - Brick
T - Tank
E - Eagle (headquarters)

Task

Implement method **void fire()**.

When called tank should produce new bullet.
This bullet should smoothly move to the opposite side.

Bullet should be destroyed when rich the opposite side.

Ignore all the objects on battle field for now.

Task

Implement method **boolean checkAndProcessInterception()**.

Should return true if bullet located in non-empty quadrant.

Also update your **void fire()** method.

When the bullet shoot something method should clean appropriate quadrant and destroy the bullet.

Final Task

Old friend Nikolay said that he just opened a shop where he is selling birds and you immediately gave him an advice to use a computer program.

You explained that you are a programmer and you will write this program for Nikolay right now.

Final Task, continue

Nikolay said that he has some notes and showed you the next:

- Bird name (crow, eagle, duck)
- Current price in USD (crow - 1, eagle - 5, duck - 0.5)
- Count of birds in the store (crow - 20, eagle - 23, duck - 4)
- Birds sold (crow - 3, eagle - 1, duck - 23)

Final Task, continue

Implement your programm in file **Birds**.

Implement next methods:

- how many birds sold
- how many birds of given type in stock
- what types of birds less then 3 in stock

Make this program useful for your friend !!!

Tasks (optional)

- Exercise 1

Home Work (Optional)

Implement method **void randomMove()**.

One method call should smoothly move the tank one quarant to random direction.

Do not use **java.util.Random**

Home Work (Optional)

Implement method
void moveToQuadrant(String quadrant).

Where **quadrant** - quadrant name in format **a4**, **h6**, **i8**.

Tank should detect given quadrant and smoothly move to it.

Task (Optional)

Implement method **void clean()**.

When method called tank should destroy all the objects on battle field in less then 30 seconds.