

Terminology of a method

- **Goal:** helper method that can **draw a random integer between start and end** (inclusive)

access modifier return type parameters / arguments

```
public static int getRandomNum(int start, int end)
{
    return (int) (Math.random() *
                (end - start + 1)) + start;
}
```

return statement

method name

Naming convention: start lowercase, uppercase each new word

Array parameters

- Arrays can be passed as arguments

```
public class AverageArray
{
    public static double average(int [] nums)
    {
        long total = 0;
        for (int i = 0; i < nums.length; i++)
            total += nums[i];
        return (double) total / (double) nums.length;
    }

    public static void main(String [] args)
    {
        int [] vals = new int[1000];
        for (int i = 0; i < vals.length; i++)
            vals[i] = RandomUtil.getRandomNum(1, 10);
        System.out.println("avg " + average(vals));
    }
}
```

```
% java AverageArray
avg 5.508
```

Array as a return value

- Arrays can be returned from methods
 - Method must create and fill in the array

```
public class ReturnArray
{
    public static double [] getRandomArray(int N)
    {
        double [] result = new double[N];
        for (int i = 0; i < N; i++)
            result[i] = Math.random();
        return result;
    }

    public static void main(String [] args)
    {
        double [] randNums = getRandomArray(Integer.parseInt(args[0]));
        for (int i = 0; i < randNums.length; i++)
            System.out.println("randNums[" + i + "] = " + randNums[i]);
    }
}
```

Pass by value

- Java passes parameters by value (by copy)

- Changes to primitive type parameters do not persist after method returns

- Primitive types: int, double, char, long, boolean

```
public static int sum(int a, int b)
{
    int result = a + b;
    a = 0;
    b = 0;
    return result;
}
```

```
% java PassByVal
```

```
sum = 5
```

```
c = 2
```

```
d = 3
```

```
int c = 2;
int d = 3;
System.out.println("sum = " + sum(c, d));
System.out.println("c = " + c);
System.out.println("d = " + d);
```

Pass by value

- Java passes parameters by value (by copy)
 - Changes to *immutable* String type parameters do not persist after the method

```
public class StringPuzzler
{
    public static void printStuff(String word)
    {
        word = "best";
        System.out.print(word);
        word = "bestest";
    }

    public static void main(String [] args)
    {
        String word = "worst";
        System.out.print("It was the ");
        printStuff(word);
        System.out.print(" " + word + " of times");
    }
}
```

```
% java StringPuzzler
It was the best worst of times
```

Changing an array parameter

- Methods can change *elements* of passed array
 - Changes DO persist after the method

```
public class ArrayClearer
{
    public static void clear(int [] nums)
    {
        for (int i = 0; i < nums.length; i++)
            nums[i] = 0;
    }

    public static void main(String [] args)
    {
        int [] scores = {90, 85, 99, 45};
        for (int i = 0; i < scores.length; i++)
            System.out.print(scores[i] + " ");
        System.out.println();
        clear(scores);
        for (int i = 0; i < scores.length; i++)
            System.out.print(scores[i] + " ");
    }
}
```

```
% java ArrayClearer
90 85 99 45
0 0 0 0
```

Java primitive types (plus String)

Java type	what it stores	examples	default value
byte	tiny integer values -128 to 127	3 -87	0
short	small integer values -32,768 to 32,767	-3433 123	0
int	integer values -2,147,483,648 to 2,147,483,647	42 1234	0
long	big integer values -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	5454 -43984938	0
double	floating-point values	9.95 3.0e8	0.0
float	less precise floating-point values	9.95f 3.0e8f	0.0
boolean	truth values	true false	false
char	characters	'a', 'b', '!'	'\u0000'
String	Sequences of characters	"Hello world!"	null

Equality: integer primitives

- **Boolean operator ==**
 - See if two variables are exactly equal
 - i.e. they have identical bit patterns
- **Boolean operator !=**
 - See if two variables are NOT equal
 - i.e. they have different bit patterns

```
int a = 5;

if (a == 5)
    System.out.println("yep it's 5!");

while (a != 0)
    a--;
```

This is a safe comparison since we are using an integer type.

Equality: floating-point primitives

- Floating-point primitives
 - i.e. double and float
 - Only an approximation of the number
 - Use == and != at your own peril

```
double a = 0.1 + 0.1 + 0.1;
double b = 0.1 + 0.1;
double c = 0.0;

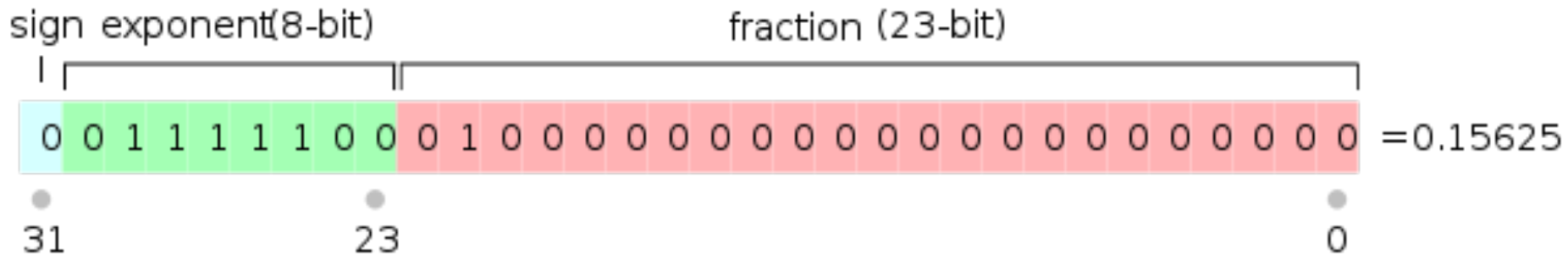
if (a == 0.3)
    System.out.println("a is 0.3!");

if (b == 0.2)
    System.out.println("b is 0.2!");

if (c == 0.0)
    System.out.println("c is 0.0!");
```

Equality: floating-point primitives

- Floating-point primitives
 - i.e. double and float
 - Only **an approximation** of the number



Floating-point numbers are shoved into a binary number.
32 bits for a float, 64-bits for a double

Equality: floating-point primitives

- Floating-point primitives

- i.e. double and float

- Only **an approximation** of the number

- Use **==** and **!=** at your own peril

```
double a = 0.1 + 0.1 + 0.1;
double b = 0.1 + 0.1;
double c = 0.0;

if (a == 0.3)
    System.out.println("a is 0.3!");

if (b == 0.2)
    System.out.println("b is 0.2!");

if (c == 0.0)
    System.out.println("c is 0.0!");
```

This works as long as no calculation has been done on 0.0 value and both are typed double.

```
b is 0.2!
c is 0.0!
```

Safe floating-point equality check

- Floating-point primitives

- Check if sufficiently close to target value

```
double a = 0.1 + 0.1 + 0.1;
double b = 0.1 + 0.1;
double c = 0.0;
final double EPSILON = 1e-10;

if (Math.abs(a - 0.3) < EPSILON)
    System.out.println("a is 0.3!");

if (Math.abs(b - 0.2) < EPSILON)
    System.out.println("b is 0.2!");

if (c == 0.0)
    System.out.println("c is 0.0!");
```

```
a is 0.3!
b is 0.2!
c is 0.0!
```

Summary

- **Static methods**
 - Can take arrays as parameters
 - Can return an array as a result
 - Method is responsible for creating and filling array
- **Changing parameters in a method**
 - Changes to primitive types do not persist
 - Changes to String types do not persist
 - Changes to elements of arrays *DO* persist
- **Primitive types have default values**
 - Watch out for String null default