EPAM JavaLab

Basic syntax

Naming conventions

Package

```
package com.epam.lab.droids;
```

Methods

```
void destroyEnemy(Droid enemy)
```

Variables

```
double energyLevel;
```

Classes & Interfaces

```
class BattleDroid
```

Constants

```
final static int MAX_DROIDS_IN_SQUAD;
```

Keywords

synchronized	package	interface	double	abstract
this	private	if	default	boolean
throw	protected	implements	do	byte
throws	public	import	extends	break
transient	return	instanceof	else	char
try	short	int	false	case
true	static	long	final	catch
void	strictfp	null	finally	class
volatile	super	native	float	continue
while	switch	new	for	interface@

Identifiers

```
JediKnight

jediName

jedi_name

_R2D2_port

$strangeVar
```

```
JediKnight ≠ jediKnight
```

Comments

```
/* C style */
// C++ style
/** Javadoc */
```

Self Documented

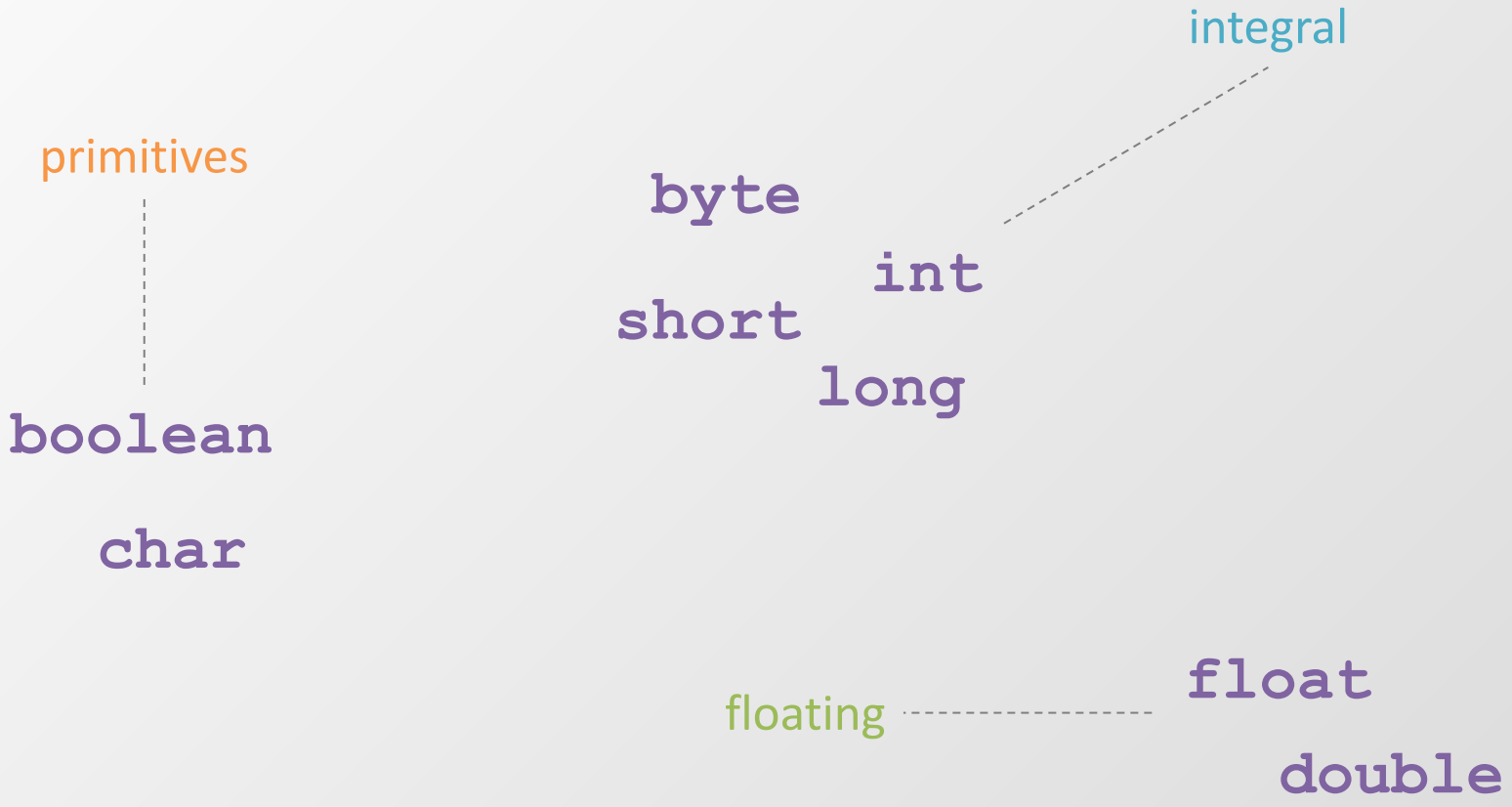
Statement

```
String a = "" + b + 32 + droid.getEnergy();
```

Blocks

```
public void doSomething(int b){
 String a = "" + b + 32 + droid.getEnergy();
}
```

Primitives



Primitives

```
int decVal = 26; // The number 26, in decimal
int octVal = 032; // The number 26, in octal
int hexVal = 0x1a; // The number 26, in hexadecimal
long longVal = 5L;
short shortVal = 4s;
```

integral

```
float floatVal = 5.4f;
double doubleVal = 0.32;
```

floating

Primitives vs. Wrappers

```
byte   -> Byte
short   -> Short
int   -> Integer
long   -> Long
float   -> Float
double   -> Double
```

References

```
Droid c3po = new TranslationDroid();
c3po.translate(text);
```

~~*c3po;~~

~~&c3po;~~

References vs primitives

```
public class Droid{

    private int energy = 0; //energy is a primitive.
    private Blaster blaster;// blaster is a null reference
                    //to a Blaster object.

    public Droid(int power, int speed, int energy)    {
        this.energy = energy;
        blaster = new Blaster(power, speed);
        // blaster is now                          // initialized
and points                              // to the Blaster object
                  // located on the heap.
     }

     public static void main(String args[ ]){
        Droid droid = new Droid(3,6,5);// droid is a reference.
        …
     }
    …
}
```

Instantiation of an object

```
Droid c3po = new TranslationDroid();
```

Expressions

Operators

```
expr++ expr--
++expr --expr +expr -expr
* / %
+ -
<< >> >>>
< > <= >= instanceof
== !=
?:
= *= /= %= += -= <<= >>= >>>= &= ^= |=
```

Bits operators

```
| // or
^ // xor
& // and
~ // inversion
```

Logical operators

```
! // not
|| // or
&& // and
```

Strings

```
String s = “Hello ”;
String name = “Skywalker”;
int num = 2;
s = s + ”to “ + name + ” and his “ +
    + num + ” droids.”;
System.out.println(s);

/* “Hello to Skywalker and his 2 droids.”
   will be printed.  */
```

If - else

```
if (droidsAmount > MAX_DROIDS_IN_SQUAD)
  createAnotherSquad();
else if (droidsAmount < MIN_DROIDS_IN_SQUAD)
  dismissSquad();
else
  deploySquad();
```

Switch

```
switch (expr1) {

   case constant2:
      //statements
      break;
   case constant3:
      //statements
      break;
   default:
      //statements
      break;

}
```

Loops - for

```
for(int i=0; i<5; i++){
   //do something
}
```

Loops - foreach

```
for(Droid enemyDroid : enemyDroidsList)
   //do something with enemy droid
}
```

Loops - while

```
while(droidsAmount > 3){
   squad.attack(enemy);
}
```

Loops - do - while

```
do{
   squad.attack(enemy);
}while(droidsAmount > 3)
```

Loops - flow control

```
break
continue
return
label:
```

Q&A

PRACTICE

Task

1. Compile and run java app from console.
2. Write program, which will pass requirements:
   - User enter the interval (for example: [1;100]);
   - Program prints odd numbers from start to the end of interval and even from end to start;
   - Program prints the sum of odd and even numbers;
   - Program build Fibonacci numbers: F1 will be the biggest odd number and F2 – the biggest even number, user can enter the size of set (N);
   - Program prints percentage of odd and even Fibonacci numbers;
3. **Object-Oriented analysis and design.**