KCL ROBOTICS

LEGO ROBOTIC ARM WORKSHOP

SOFTWARE NEEDED

SOFTWARE NEEDED

Java Development Kit

Java Development Kit (JDK) is an application that is used to create programs using the **Java** programming language.

In order to program the LEGO NXT Bricks, you will need the 32-bit version for all of the software used, including JDK.

Link:
http://download.oracle.com/otn-pub/java/jdk/8u121-b13/e9e7ea248e2c4826b92b3f075a80e441/jdk-8u121-windows-i586.exe

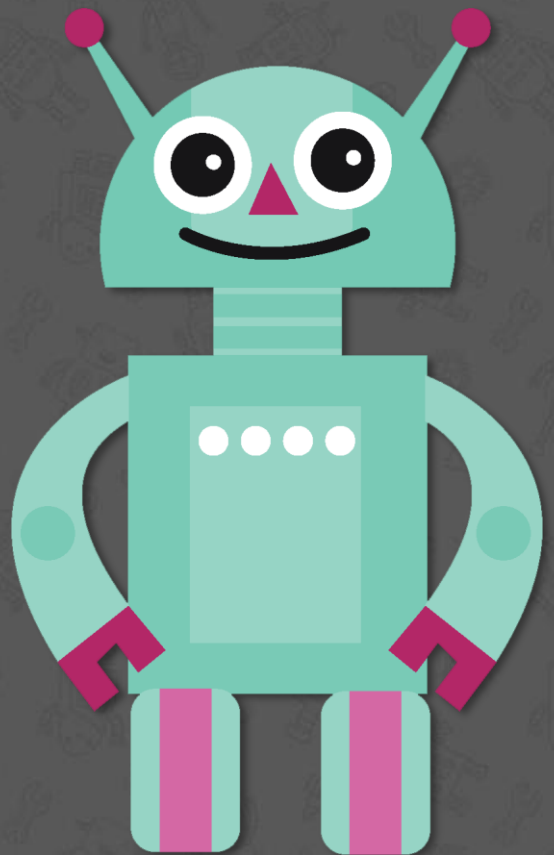
SOFTWARE NEEDED

ECLIPSE

Eclipse for Java is an IDE (Integrated Development Environment) that can be used in order to code in the Java programming language.

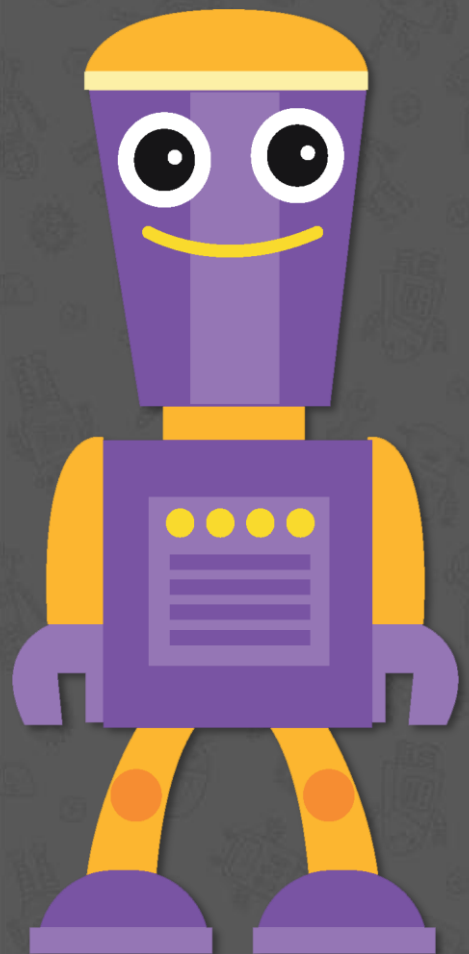
Link:

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/neon/2/eclipse-java-neon-2-win32.zip



SOFTWARE NEEDED

NXT FANTOM DRIVER



The **NXT Fantom Driver** is the official software from LEGO that permits the NXT Brick to be recognized by the system.

Link:

https://lc-www-live-s.legocdn.com/r/www/r/mindstorms/-/media/franchises/mindstorms%202014/downloads/firmware%20and%20software/nxt%20software/nxt%20fantom%20drivers%20v120.zip?l.r2=-964392510

SOFTWARE NEEDED

LEJOS

LEJOS (LEGO Java Operating System) is a tool that handles programming the NXT with the Java language.

Link:

https://sourceforge.net/projects/nxt.lejos.p/files/0.9.1beta-3/leJOS_NXJ_0.9.1beta-3_win32_setup.exe/download

It needs to be also installed in **Eclipse** (Go to Help -> Install new software -> Paste this link http://www.lejos.org/tools/eclipse/plugin/nxj/).

PROGRAMMING CONCEPTS

PROGRAMMING CONCEPTS

CLASSES

In order to write code, you need a file to store it inside. These files are called **classes**, and have the extension ".java". (e.g. Main.java)

Think of a class as a normal **text file**, with the difference that it keeps code inside, not just normal text.

The first step in writing Java in a class is to declare the class, which has the following syntax:

```
public class FileName { // code here }
```

Aim to capitalize the first letter, as a convention.

PROGRAMMING CONCEPTS

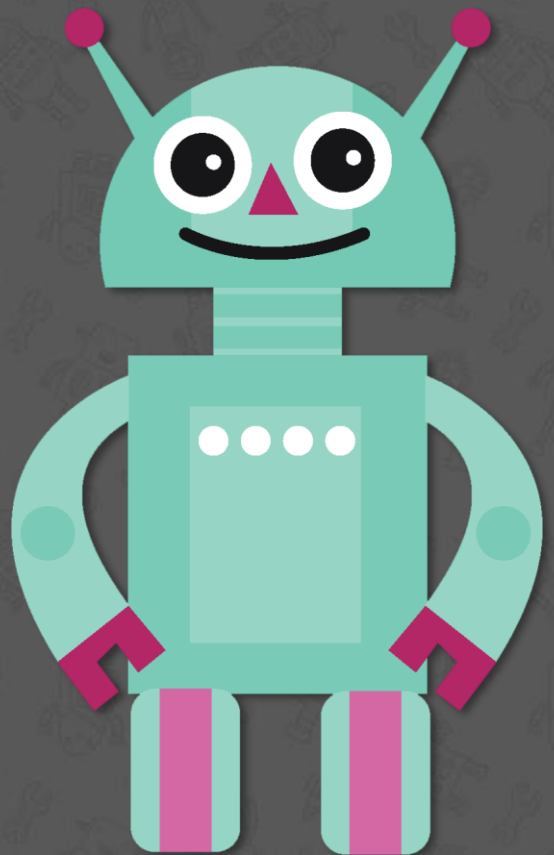
BRACKETS AND COMMENTS

We declared a class, and then we added **brackets**. We use **brackets** to ensure that we know where a code sequence starts and ends.

Comments are not part of the code, they are just ignored. We use **comments** to describe what the code does.

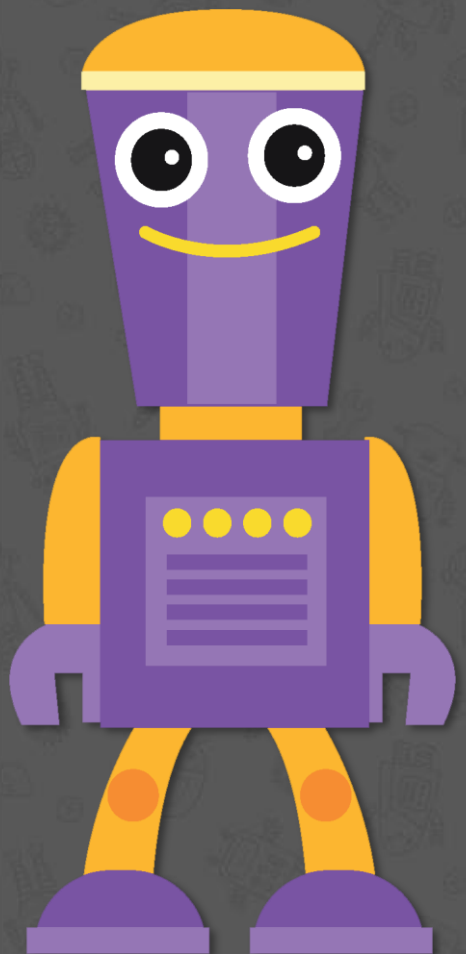
There are **two** types of comments:

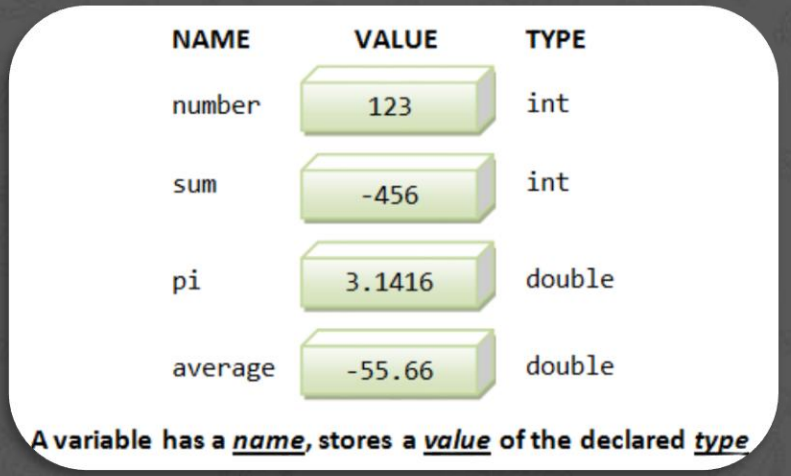
Single-line comments are delimited by “//”, there are also **multi-line comments**, which start with “/*” and end with “*/”.



PROGRAMMING CONCEPTS

VARIABLES





Very often in a program, you want to store an information (e.g. the sum of two numbers, the speed of a motor). This is the purpose of **variables**.

Java offers plenty of variable types, we will only study the following:

int – stores integers (whole numbers), e.g. int y = 2;
double – stores floating point numbers, e.g. double x = 3.5;
boolean – stores “true” or “false”, good for decisions, e.g. boolean ok = true;

PROGRAMMING CONCEPTS

METHODS

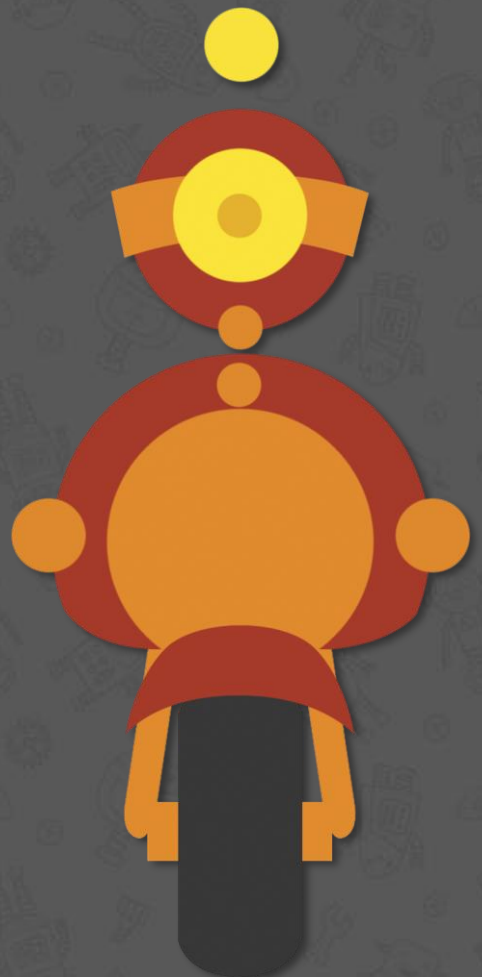
After declaring a **class**, your program needs to know what its purpose is. For explaining how our program should work, we use **methods** inside the class.

Every program needs to have a **main method**, which is always executed. The syntax for the main method might seem complicated at first, but it will gradually start to make sense.

```
public static void main(String[] args) {
        // code here
}
```

PROGRAMMING CONCEPTS

CODE SAMPLE AND PRINT METHOD



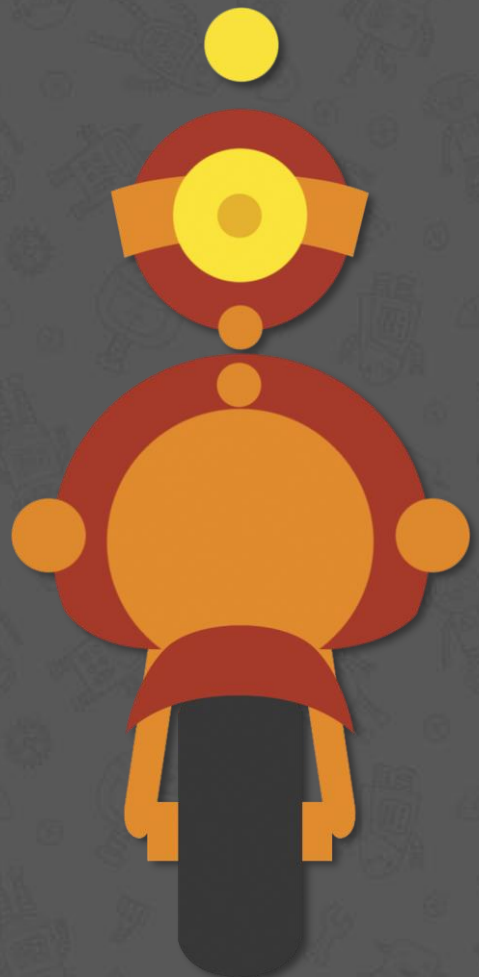
```
public class ShowNumber {
        public static void main(String[] args) {
                int var = 10; // declare and initialize “var” with 10.
                var = var + 25; // add 25 to the old value of “var”
                System.out.println(“val is” + var); // prints “val is 35”
        }
}
```

The method used for printing in the console is “System.out.println(text);”. Combining **text** (denoted by “ “) with **variables** is done by using the “+” operator, that does concatenation, not the usual addition.

EXERCISES

EXERCISES

PRACTICE MAKES PERFECT



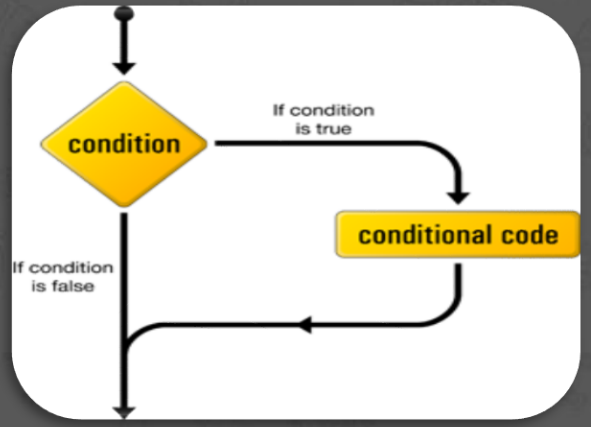
In order to create a **class**, open Eclipse and go to File -> New Project -> Other -> LEJOS, and make sure you tick the option that includes the main method.

1. Print your name to the console (easy)
2. Declare a variable, increment it by 2, and print it (easy)

CONTROL FLOW

CONTROL FLOW

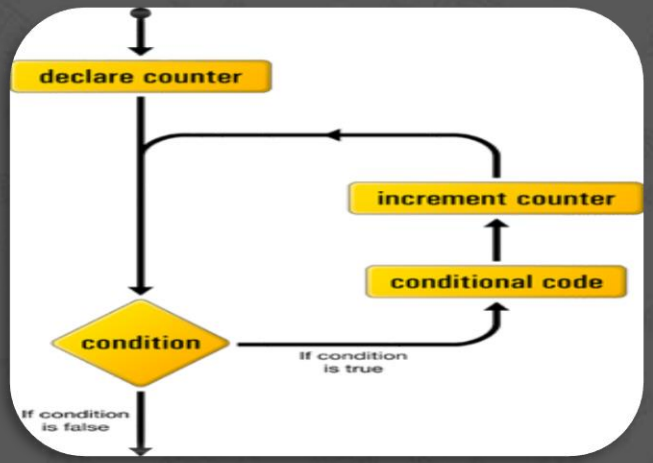
IF STATEMENTS



When building programs, an important part is to make **decisions**. This is why we use **control flow**.

"if" statements are very useful when you want your program to do different tasks depending on its state. They are usually followed by an **"else"**, which handles the situation that your condition is not true.

e.g.
```
int var = 5;
    if(var > 5) {
        System.out.println("Bigger than 5");
    }
    else {
        System.out.println("Not bigger than 5");
    }
```



CONTROL FLOW

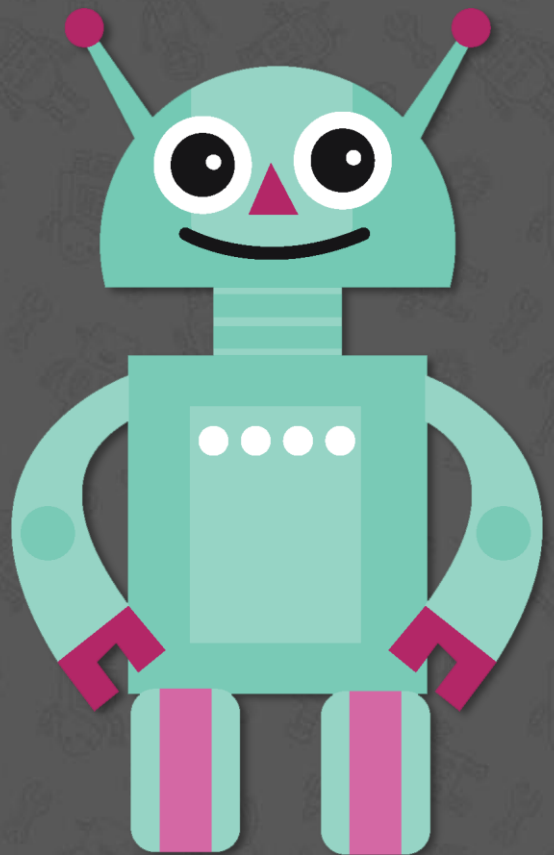
FOR LOOPS

“For” loops are used to make the code repeat for a *known* number of times.

Java syntax for “for”:

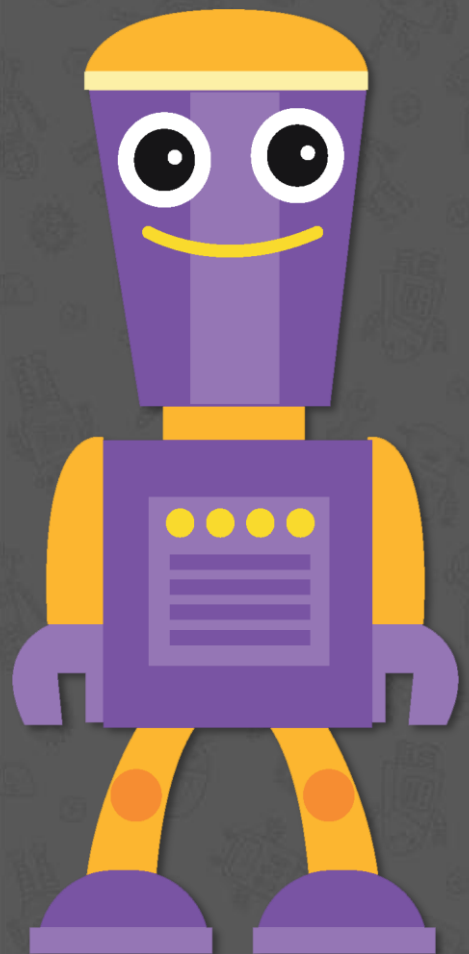
for(initialize variabile; condition on variable; update variable)

e.g.
```
for(int index = 0; index < 5; index = index + 1) {
        System.out.println(“I will be printed 5 times”);
    }
```



CONTROL FLOW

UPDATING VARIABLES

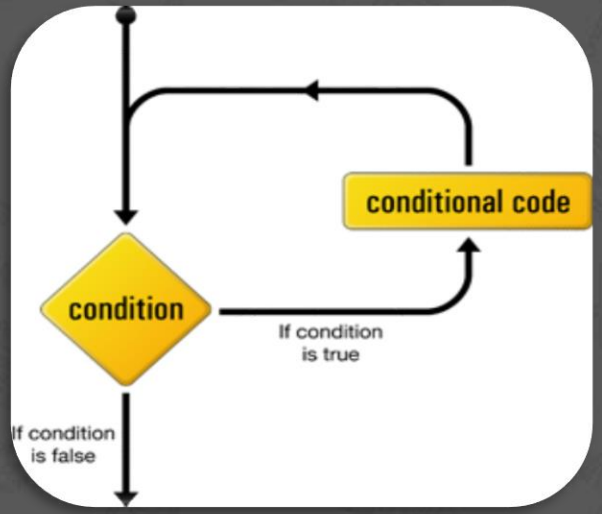


It is correct to write “index = index + 1”, but it is too long and Java offers simpler methods to do this.

You can use the operator **“+=“**, e.g. “index = index + 2” is equivalent to “index += 2” (**+** is just a case, it works for all mathematical operators: **“-=“**, **“*=“**, **“/=“**, **“%=“**)

Because it is often to need to increment/decrement a variable by only one (frequent in “for” loops), you can also use **“++”** and **“--”** operators.

A normal “for” definition looks like this: for(int i=0; i<5; i++) { }



CONTROL FLOW

WHILE LOOPS

“While” loops are used to make the code repeat for an *unknown* or *known* number of times.

Java syntax for “while”:

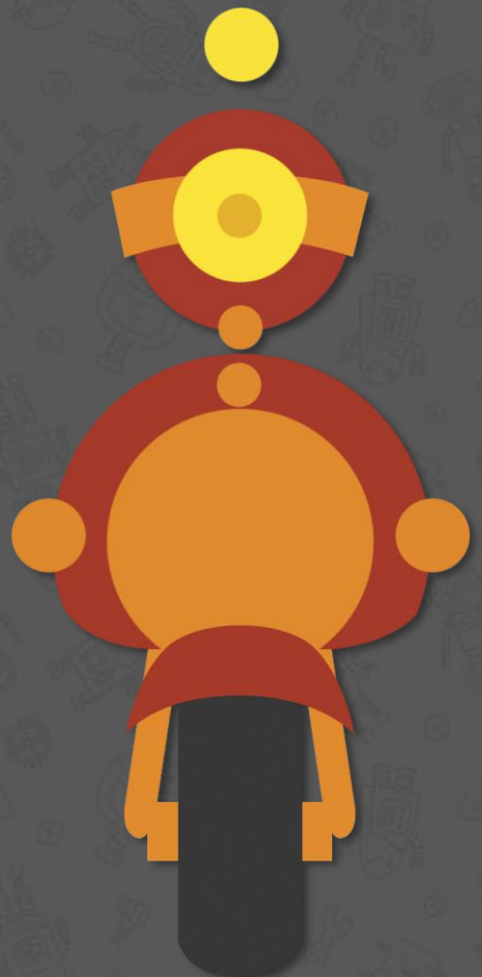
```
while(condition) { }
```

e.g.

```
while(true) { // I will never stop!
    System.out.println(“Don’t stop me now!”);
}
```

CONTROL FLOW

SAMPLE CODE & COMMENTS



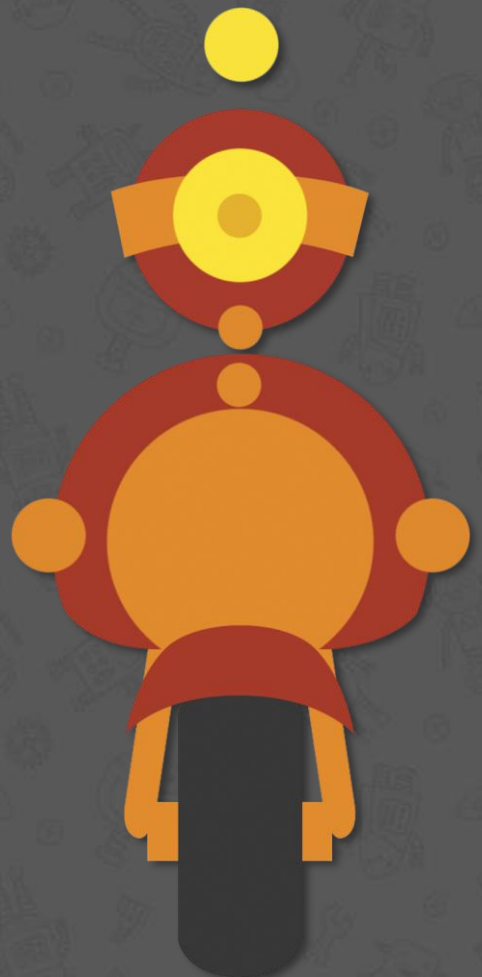
A class that checks if any number from 0 to 10 is divisible by 3.

```
/* Main class.
*/
public class Main {
        public static void main(String[] args) {
                boolean isDivisible = false; // need to have a value
                for(int i=0; i<10; i++) {
                        if(i%3 == 0) { // check divisibility for current nr
                                isDivisible = true;
                        }
                }
                System.out.println(isDivisible); // prints true
        }
}
```

EXERCISES

EXERCISES

PRACTICE MAKES PERFECT



1. Write a for loop that shows all the numbers from 0 to 20 (easy)
2. Write a for loop that shows all the even numbers from 0 to 20 (medium)
3. Write a while loop for numbers from 0 to 50 that prints checks if the last digit of the number is 5, and in that case, prints the number (hard)

OBJECT-ORIENTED PROGRAMMING

OBJECT-ORIENTED PROGRAMMING

CLASSES

A **class** is a defining concept in Object-Oriented Programming (OOP) and it can contain **methods** and **variables**.

Access modifiers that we will use are:

public – visible outside the current class
private – not visible outside the current class

Classes are always **public** (e.g. public class Car).
Methods are, in general, **public**.
Variables are **private** (they should not be seen from outside the class)

OBJECT-ORIENTED PROGRAMMING

VARIABLES

Variables can be split in two types: local variables and fields (global variables).

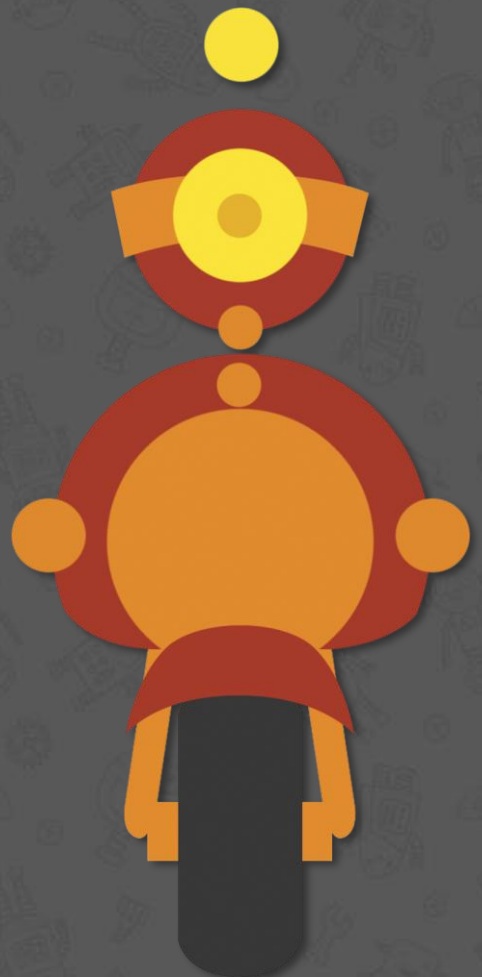
Local variables are visible only inside the method they are created in.

Fields are visible in the whole program and they also need an access modifier.

E.g. local variabile: int var; (in a method)
field: private int var; (outside any method, at the beginning of the class)

OBJECT-ORIENTED PROGRAMMING

METHODS



A **method** is a part of the code that ensures a particular functionality to the whole program.

Java syntax for a method:

private/public (static) (final) type name(parameters) { }

E.g. main method: public static void main(String[] args) { }

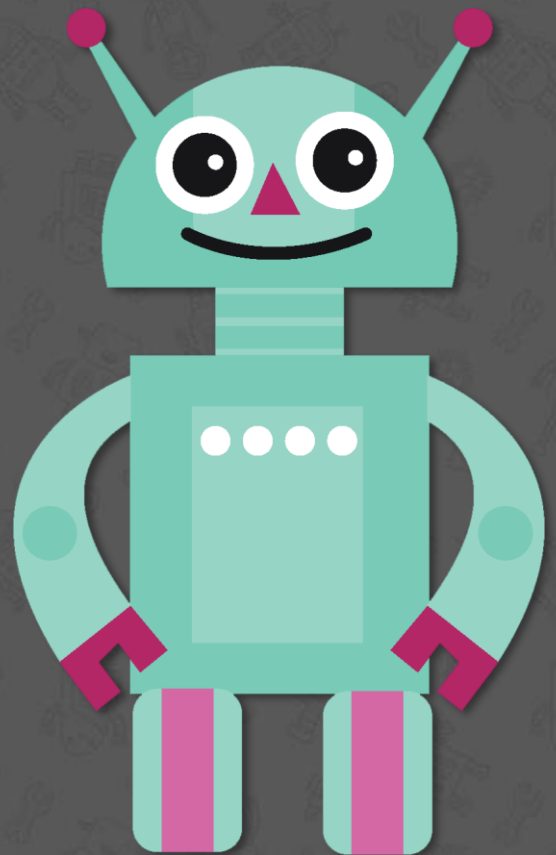
Parameters act as local variables in the method that they are in, with the difference that they are provided when calling the method from somewhere else.

OBJECT-ORIENTED PROGRAMMING

METHODS

Methods, as variables, have **types**:

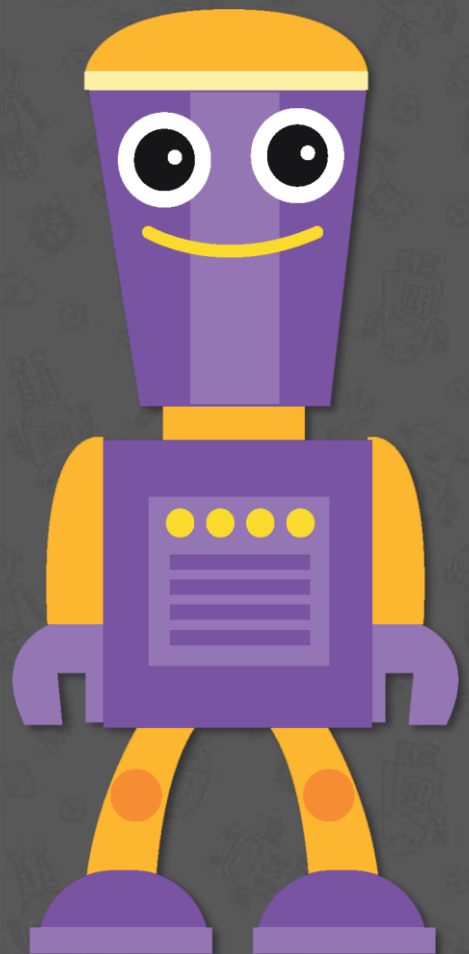
void – a method that does not return anything, acts the same as simply copying the code inside the main method.
int – a method that returns an integer (e.g. a sum)
double – a method that returns a double (e.g. returning the area of a circle)
boolean – a method that returns true or false.



A method that has a non-void type **needs** to return a value of the same type as the definition (e.g. **int** method returns an **int**). The syntax is "**return** value;".

OBJECT-ORIENTED PROGRAMMING

METHODS



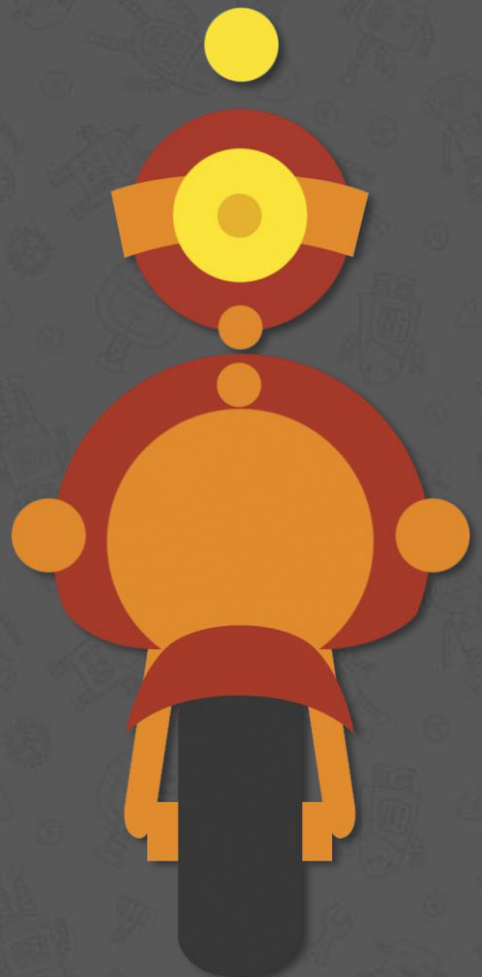
Static is a keyword which states that no matter how many instances of the same class you create, it stays the same.

You need to call **static** methods and use **static** variables inside **static** methods.

Final is a keyword that refers to something being a constant (the value can never change).

OBJECT-ORIENTED PROGRAMMING

SAMPLE CODE & COMMENTS



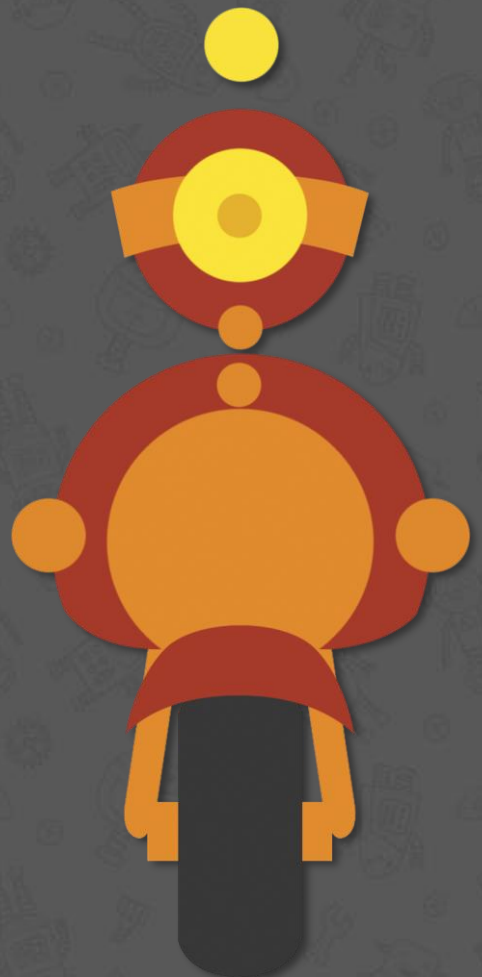
A class that calculates a sum of two numbers looks like this:

```
/* Main class.
*/
public class Main {
        public static int sum(int a, int b) {
                return a+b;
        }
        public static void main(String[] args) {
                System.out.println(sum(1,2)); // Prints 3
                System.out.println(sum(5,15)); // Prints 20
        }
}
```

EXERCISES

EXERCISES

PRACTICE MAKES PERFECT



1. Write a method that writes a given integer number as parameter to the console (easy)
2. Call the method from the main method with any number you choose (easy)
3. Write a for loop that calls your method for every number from 0 to 20 (medium)
4. Write a method that checks if a number is bigger than another given number and test it in the main method (hard)

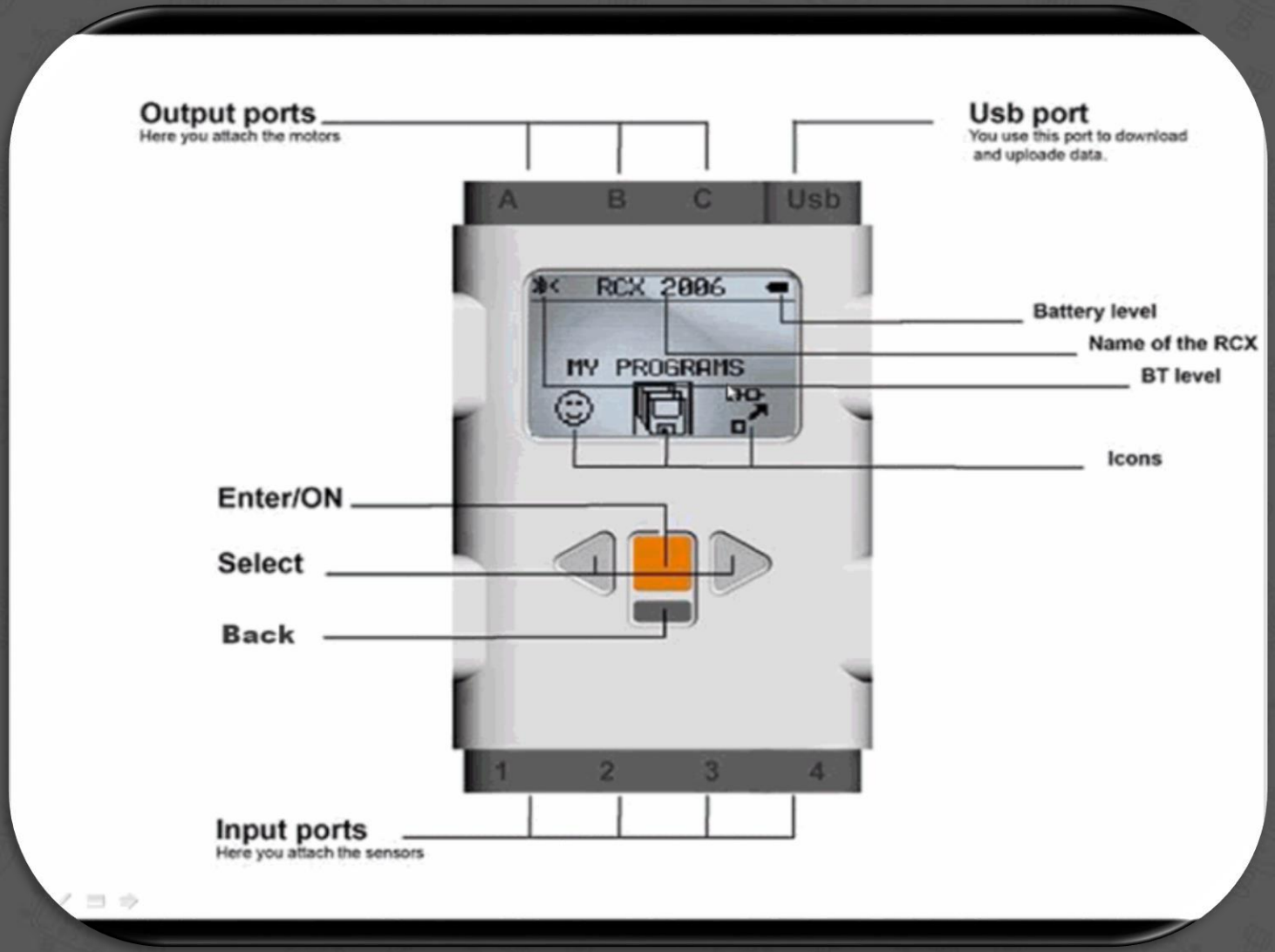
NXT ROBOTS

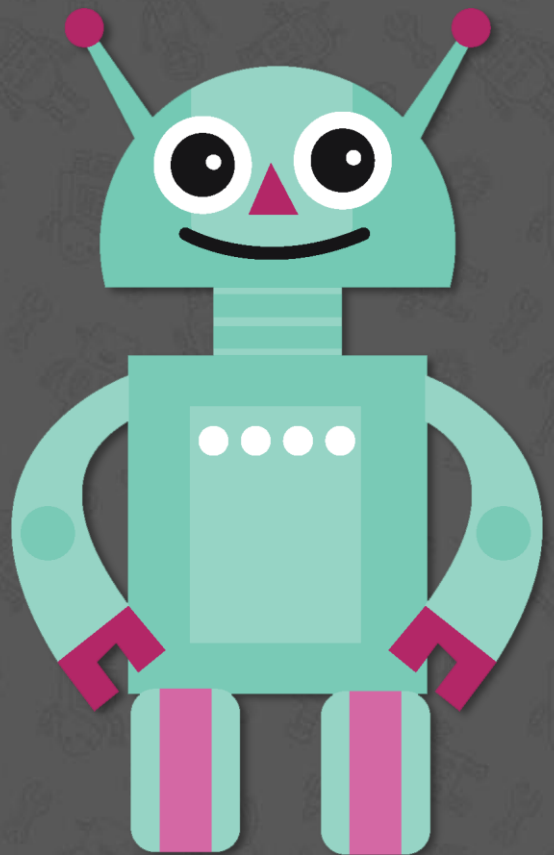
NXT ROBOTS

BRAIN OF A NXT – NXT BRICK

NXT ROBOTS

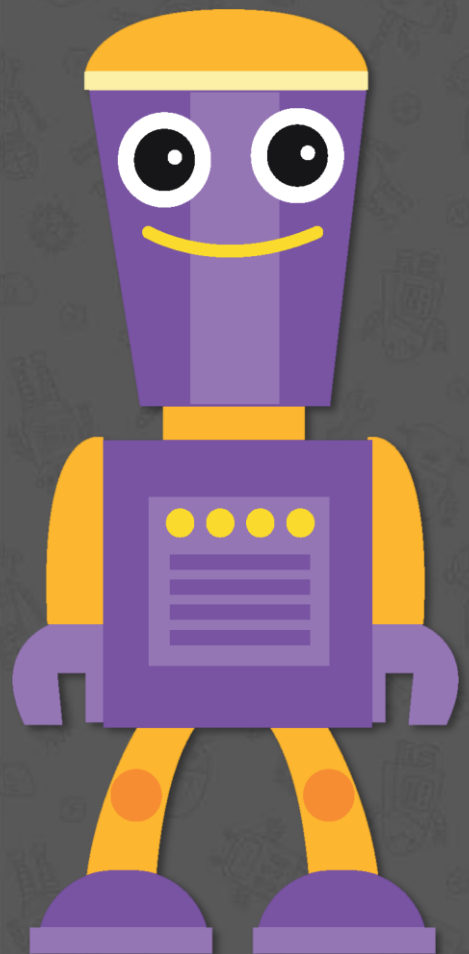
NXT BRICK





NXT ROBOTS

CONNECTIONS



Connections:

Ports A, B, C (mainly used for motors)

Ports 1-4 (usually used for sensors)

LCD display panel (you can program what to display)

4 buttons (left, right, enter, escape)

USB port (used for downloading programs to the brick)

NXT ROBOTS

ULTRASONIC SENSOR

They are used to find the distance to obstacles.

Work by transmitting ultrasonic signal, and

timing how long it takes for the signal to get back.

Works up to about 225 cm, precision of 3 cm.

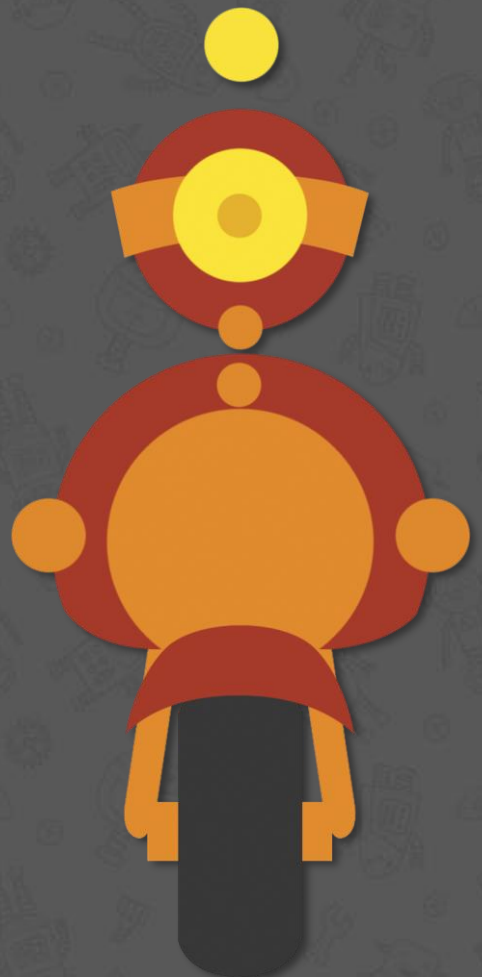
PAY ATTENTION!

Ultrasonic sensors interface with each other.

If others are using ultrasonic sensors nearby, your robot might get confused.

NXT ROBOTS

COLOUR SENSOR



They can be used to identify the **colour** of objects.

Simplest use is to detect colour patches (tracks) immediately underneath the robot.

Good for red, green, blue, white, black.

NXT ROBOTS

TOUCH SENSOR

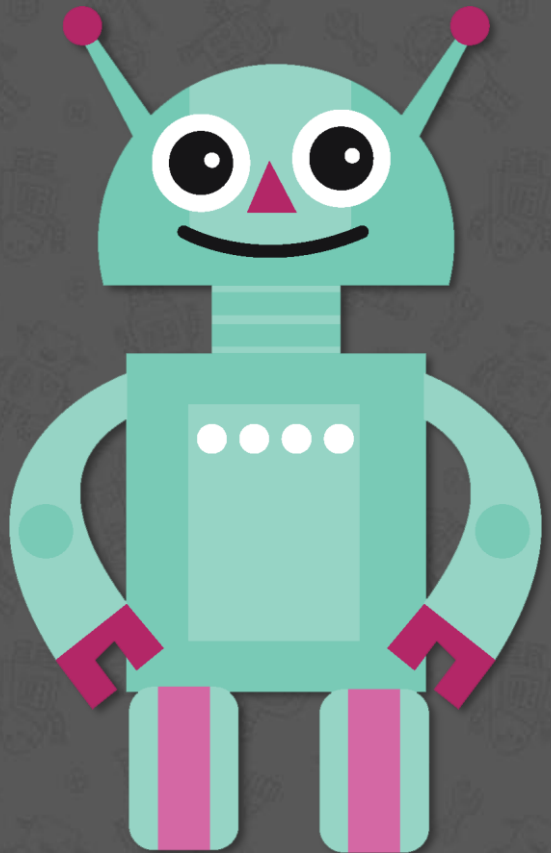
Detect sensor being pressed.

Used for the arm of the robot, to notify when the ball is taken.

Return a boolean value

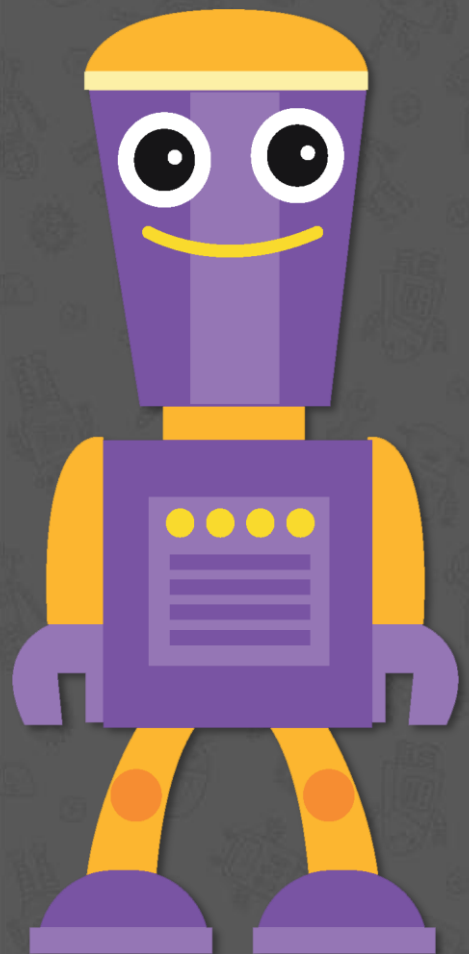
True = depressed

False = not depressed



NXT ROBOTS

SERVOMOTORS



Allow **precise positioning** of motors, fine degree of control of rotation.

Used for:

Connecting the wheels

Building the robotic arm

NXT ROBOTS

ROBOT CONFIGURATION

Output port A - robotic arm

Output port B - right wheel motor

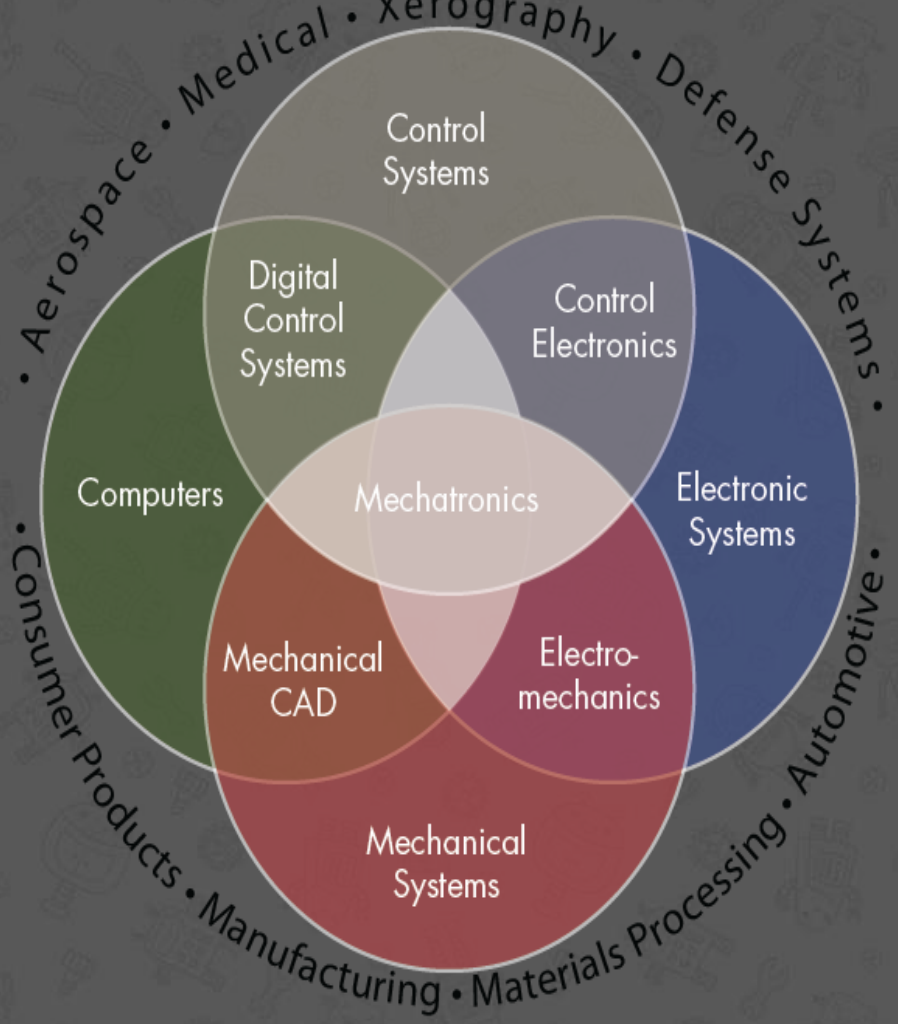
Output port C - left wheel motor

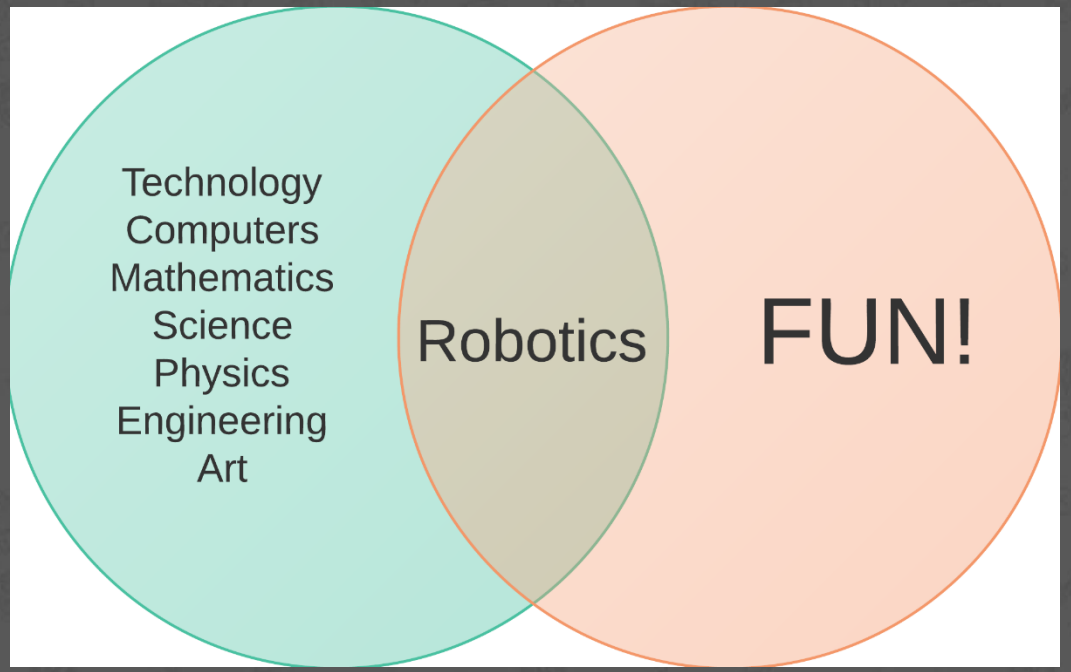
Port 1 - touch sensor

KINEMATICS

KINEMATICS

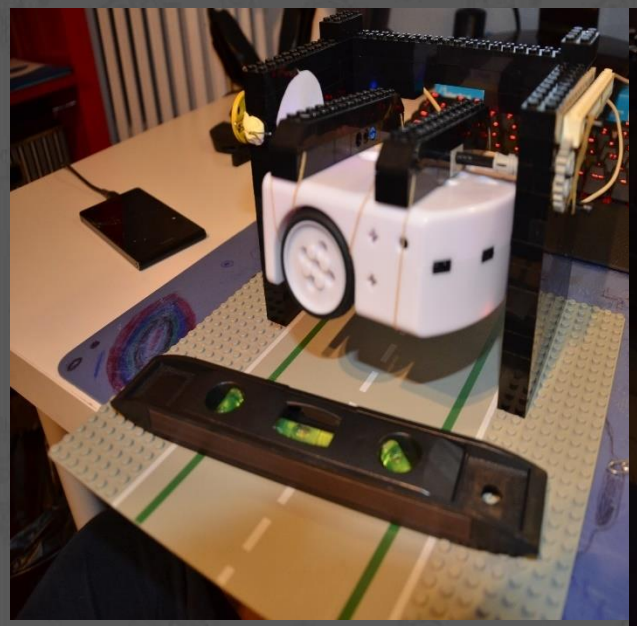
WHAT ARE ROBOTICS AND MECHATRONICS?

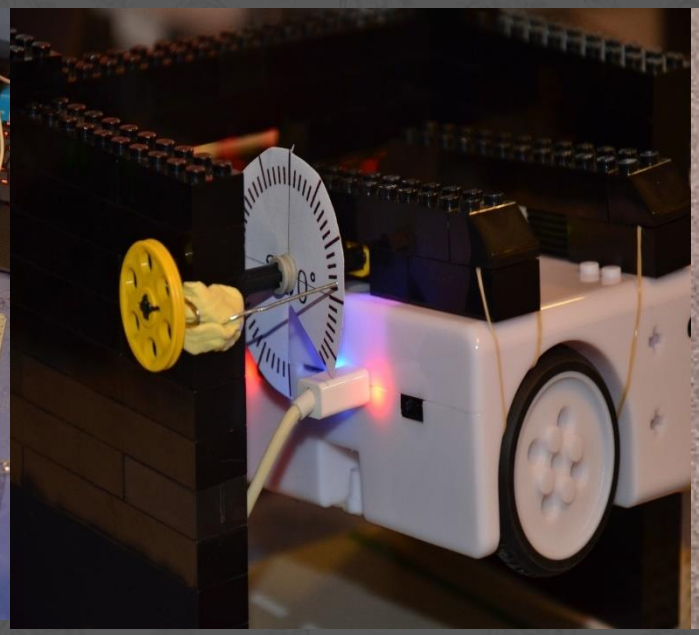


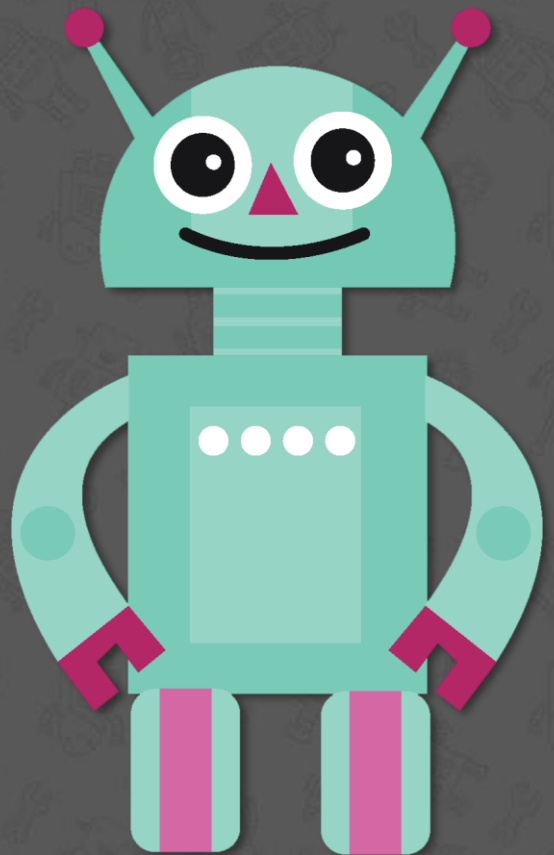


KINEMATICS

MODELLING ENVIRONMENTS

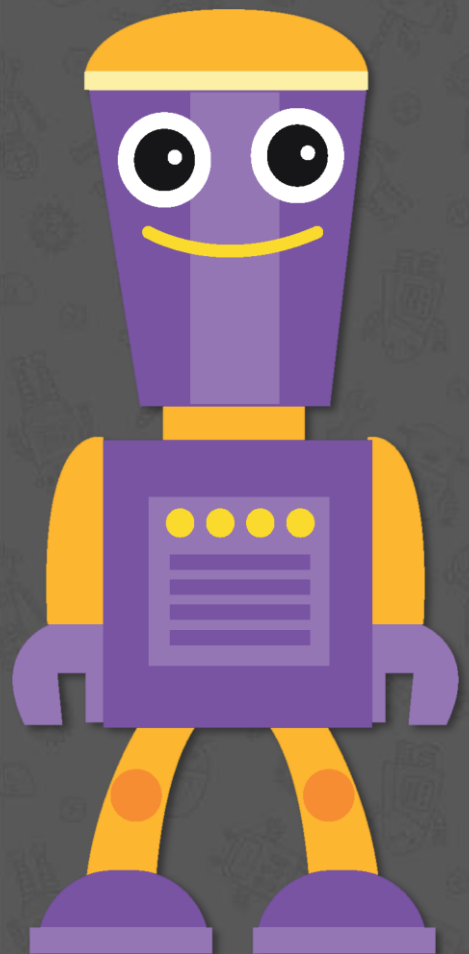






KINEMATICS

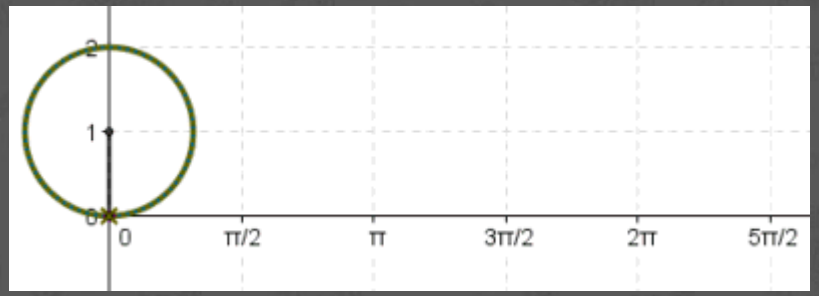
WHAT IS KINEMATICS?



“The branch of mechanics concerned with the motion of objects without reference to the forces which cause the motion.” - Wikipedia

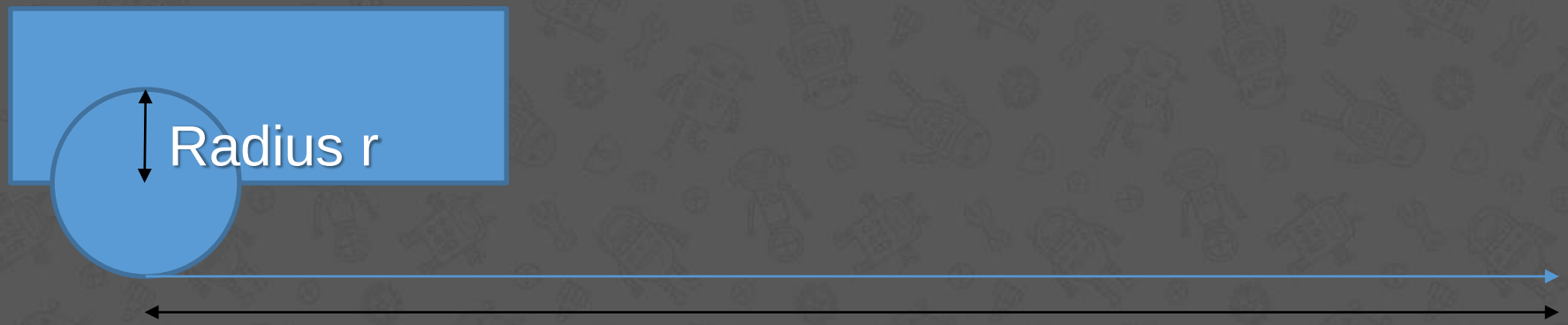
“**Forward kinematics** refers to the use of the **kinematics** equations of a robot to compute the position of the end-effector from specified values for the joint parameters” - Wikipedia

“**Inverse kinematics** makes use of the **kinematics** equations to determine the joint parameters that provide a desired position for each of the robot's end-effectors” - Wikipedia



KINEMATICS

FORWARD KINEMATICS – GOING STRAIGHT



Circumference = 2πr

Distance for 360 degree (1 full rotation) rotation of the wheel d = 2πr

Distance for 720 (2 full rotations) rotation of the wheel d = 2πr × 2

Distance for 180 (half of a full rotation) rotation of the wheel d = 2πr × 0.5

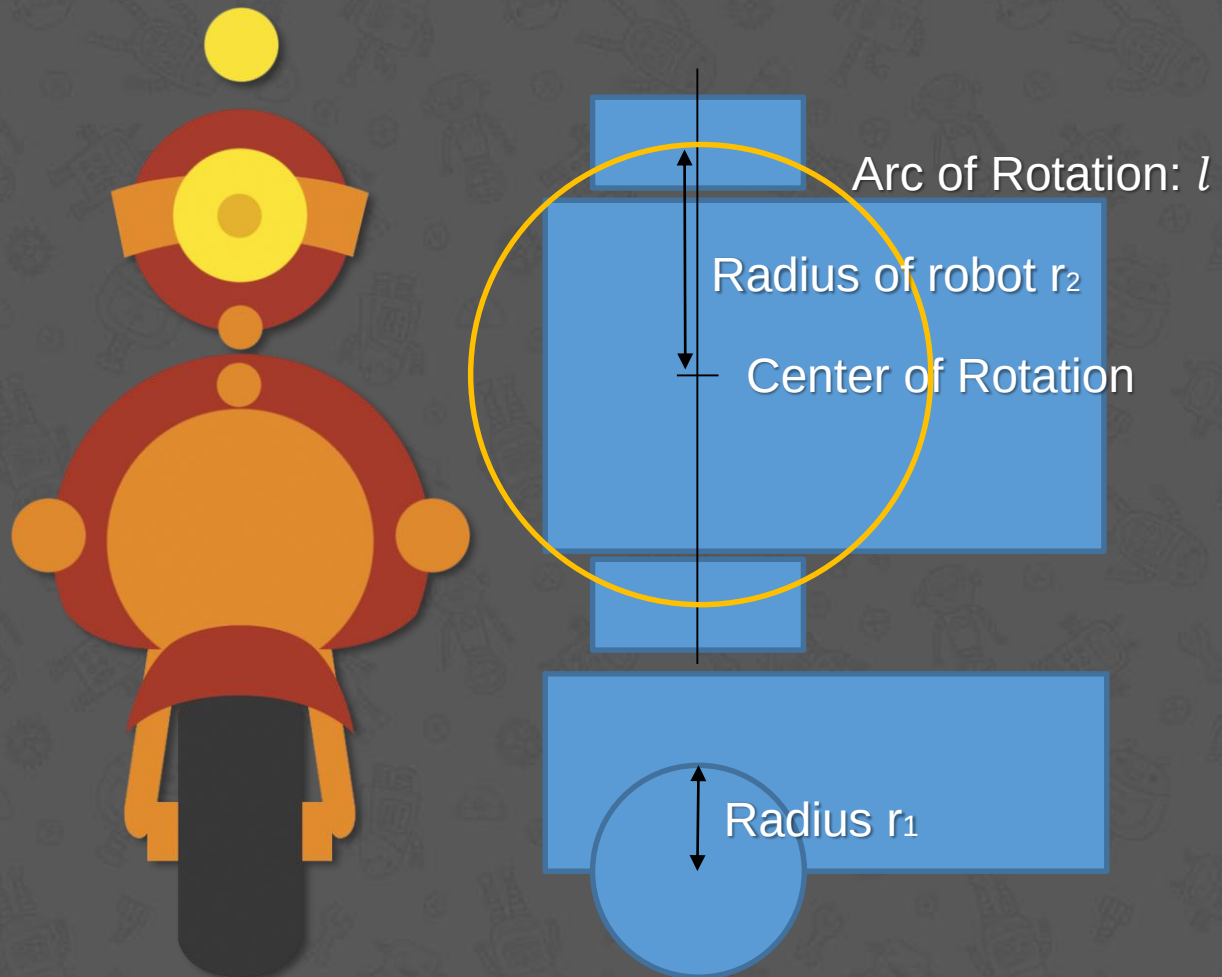
Distance for 90 (quarter of a full rotation) rotation of the wheel d = 2πr × 0.25

Ration of degrees rotated vs full rotation $= \frac{degree}{360}$

Distance Travelled d $= 2\pi r \times \frac{degree}{360}$

KINEMATICS

FORWARD KINEMATICS - TURNING



Length $l = 2\pi r_2$

Number of rotation of wheel for 360 rotation of robot:

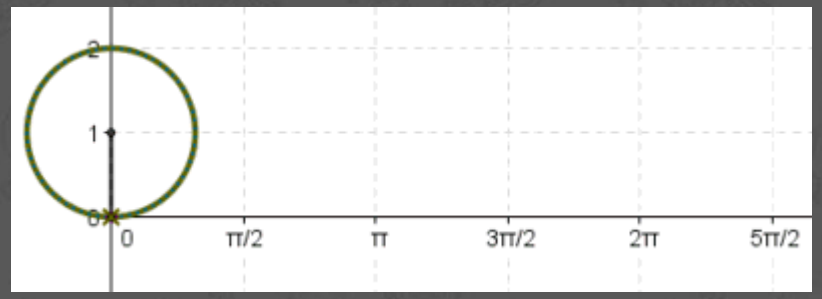
$$ratio = \frac{2\pi r_2}{2\pi r_1} = \frac{r_2}{r_1}$$

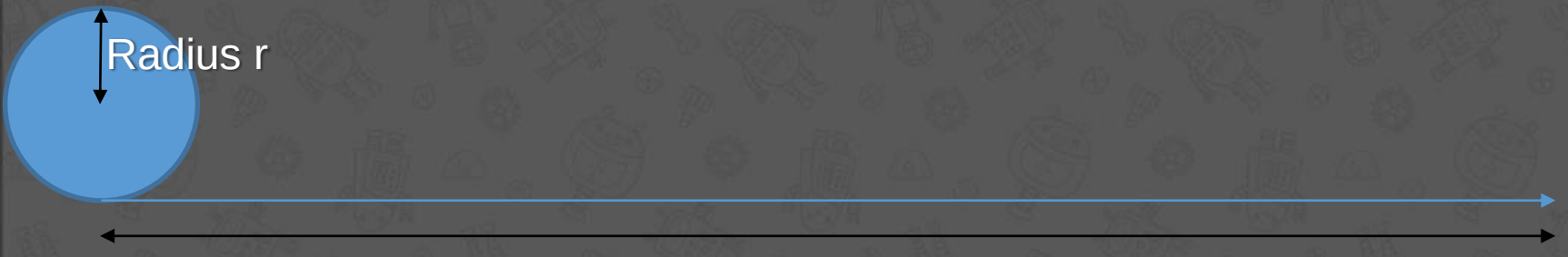
Rotation of wheel vs rotation of robot

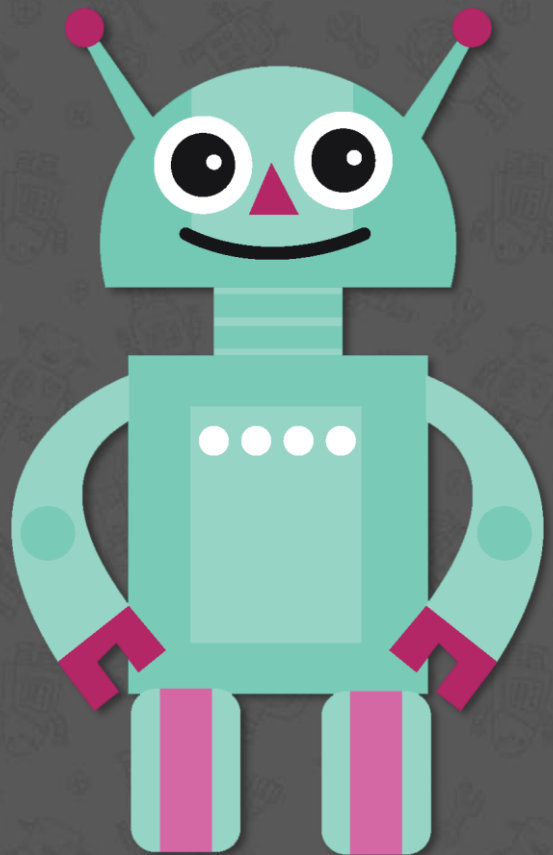
$$degrees(robot) = \frac{r_2}{r_1} \times degrees(wheel)$$

KINEMATICS

INVERSE KINEMATICS – GOING STRAIGHT







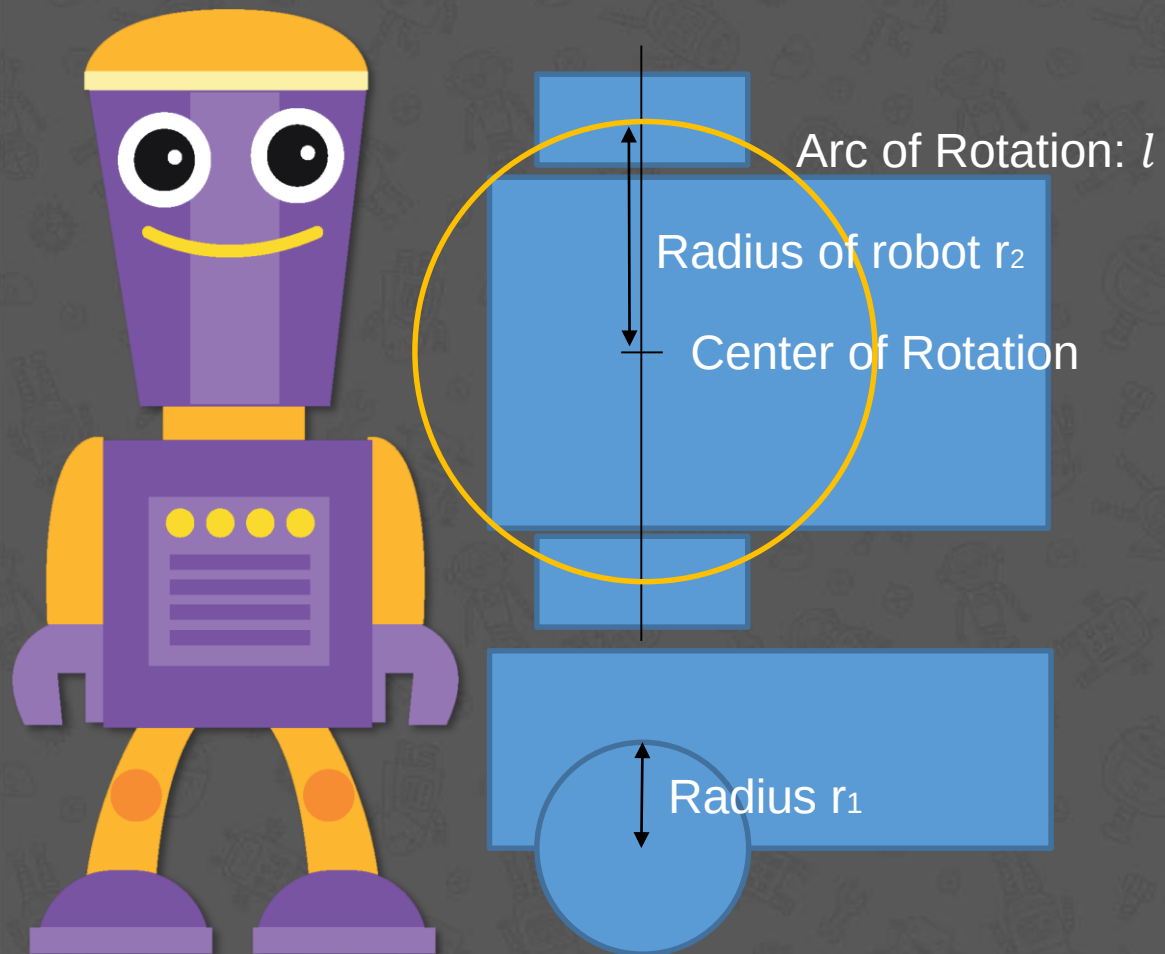
Rearrange equation from forward Kinematics to make the $degree$ the subject of the formula

$$d = 2\pi r \times \frac{degree}{360}$$

$$degree = \frac{d}{2\pi r} \times 360$$

KINEMATICS

INVERSE KINEMATICS - TURNING



Length $l = 2\pi r_2$

Number of rotation of wheel for 360 rotation of robot:

$$ratio = \frac{2\pi r_2}{2\pi r_1} = \frac{r_2}{r_1}$$

Rearrange equation to make $degrees\ rotation(wheel)$ the subject of the formula

$$degrees\ rotation\ (robot) = \frac{r_2}{r_1} \times degrees\ rotation(wheel)$$

$$degrees\ rotation\ (wheel) = \frac{r_1}{r_2} \times degrees\ rotation(robot)$$

ROBOTIC ARM API

ROBOTIC ARM API

WHAT IS AN API?

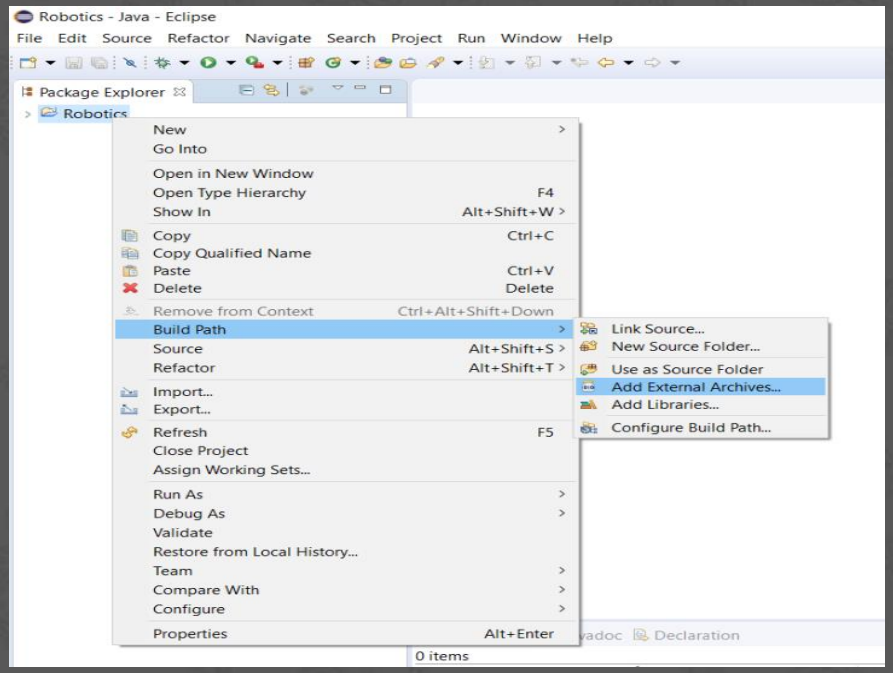
An **API** (Application Program Interface) is a ready-built resource that can be used in order to achieve some tasks.

You will be using an **API** created by our team, in order to manipulate the robot easier, without actually getting in touch with programming the parts of the robot individually.

Java **API**s are identified by the extension “.jar”.

ROBOTIC ARM API

IMPORTING THE API

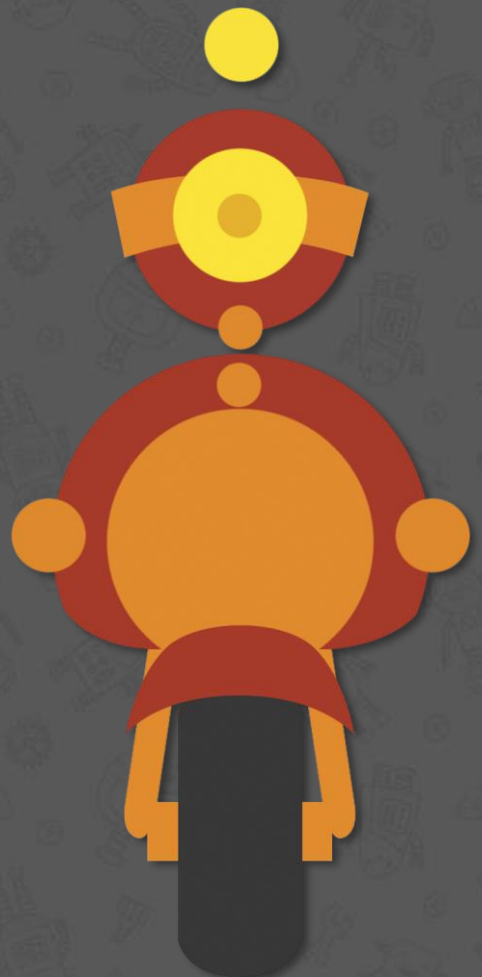


You can find below the steps for importing the API:

1. Download the .jar file (https://github.com/RoboticsKCL/LEGO-API/blob/master/RoboticArm.jar)
2. Right click on your project
3. Click Build Path
4. Click Add External Archives
5. Choose your .jar file

ROBOTIC ARM API

USING THE API



The API provides you with some methods ready to use:

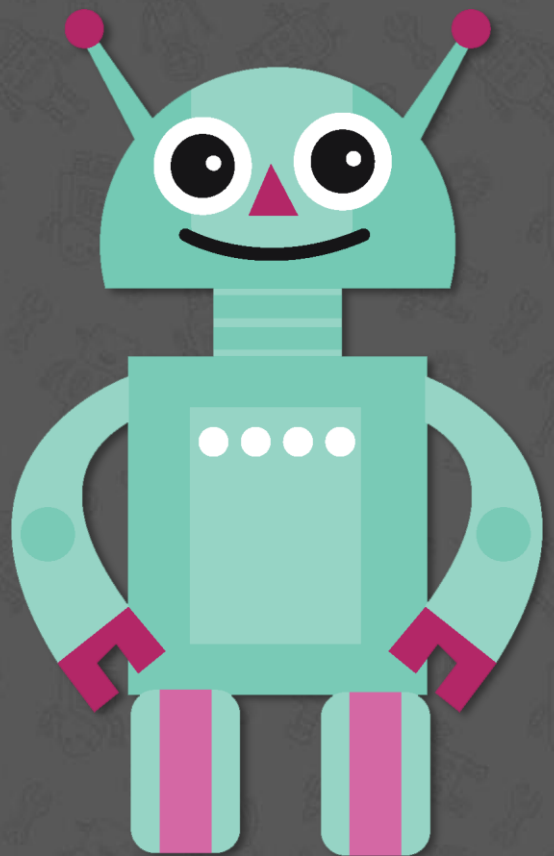
```
void moveRobotForward(int cm) // move robot forward given cm
void moveRobotBackward(int cm) // move robot backward given cm
void setWheelSpeed(int speed) // set the speed of the wheels
void setArmSpeed(int speed) // set the speed of the arm
void rotateRobotLeft(int degrees) // rotate the robot to the left
void rotateRobotRight(int degrees) // rotate the robot to the right
void catchObject()  // close the arm
void openArm() // open the arm
```

ROBOTIC ARM API

PROGRAMMING SAMPLE

A class that opens the arm and rotates the robot to the left for 90 degrees looks like this.

```
/* Main class.
*/
public class Main {
        public static void main(String[] args) {
                RoboticArm.openArm(); // open arm
                RoboticArm.rotateRobotLeft(90); // rotate left
        }
}
```



KCL ROBOTICS

THANK YOU FOR ATTENDING!