Robotics - Event Driven Programming

Dave Cohen

Term 2, Academic Year 2020/2021

Computer Science, Royal Holloway, University of London

Introduction

Groups

Deliverables

- You will work in groups of three or four. **Everyone programs.**
- You will give a group presentation at the end of module.
  Not everyone has to present - you all get the same grade.
- You will give a public robot demonstration in the summer term.
  Not everyone has to demonstrate - you all get the same grade.
- You will submit an individual report: **your own work!**.
- Your grades are very strongly affected by your participation.
- The five worksheets count towards your participation score.

Succeeding in Groups

Worksheets and Progress

- Worksheets help you to pace your progress (One+ per week).
- Every worksheet needs you to program the robot.
- You submit the code on Moodle.
- Stronger group members should run their own tutorials.
- Course leaders and/or tutors will be available at all sessions to help.
- Use the Piazza forum to ask questions
- We can help to fix robots that go bad.
- **Everyone codes.**

Projects

Kinds of Lego Robot

- Brilliant Table Toddler (maybe maps the table).
- Inspired Wall Follower (maybe maps the room).
- Clever Line Follower (maybe decides what to do at junctions).
- Sublime Object Counter (could sort objects as well).
- Maze solver, Card Player, Musical Instrument player, Intelligent Cars...
- Towers of Hanoi, ..., anything else agreed by me!

Moodle

There is a much more extensive list on Moodle.

Many groups will use Android via Bluetooth

EV3Sensors and an Android Phone Fun with Phones

The GitHub account cyclingProfessor has a repository called `EV3Sensors` that will connect successfully to an EV3 brick, and can do OpenCV image processing, find QR codes and reads NFC tags.

You must load it into a properly installed `Android Studio` on your own machine.

- You can use it as a set of extra sensors for your brick.
- For example QR, NFC, FaceFinder and Proximity Sensor
- Or a (much) better output device than the LCD on the robot.

EV3Dev Examples

The Ev3Dev-examples repository is useful: eg for the EV3Dev program for `EV3Sensors`.

Some Groups use the PC

Robot Code

The same robot code can run on the PC and control the robot. !

Then you have all of the Java ecosystem (and APIs).

This lets you use a proper Java user interface and perhaps a map on the screen etc.,

Every Group uses the Behaviour design pattern

Behaviours: Walking, talking, eating and falling

When we are *walking* about (a low-level behaviour) we can also be *eating* and *talking*.

- None of these three behaviours suppresses any of the others.
- The *eating* behaviour uses our hands.
- The *Walking* behaviour uses our legs.
- The *eating* behaviour share the mouth with *talking*

If we *fall over* this will suppress all the other three behaviours.

- It is a higher level behaviour.

We can program robots using this subsumption architecture.

Extending Behaviours: States and Personality

Changing from one state of certainty to another

- We can control the flow of behaviours using a `state` field.
- We can make a set of behaviours active using `Personality` objects.
- Different personalities can share some `Behavior` objects.
- We can even use the `state` field and behaviours to separate claps.
- Drawing a state diagram is a very useful process.

Square Dancing Robot

Its So Easy:

1. Do FOUR times:
   - 1.1 Go Forward.
   - 1.2 Turn Left

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);
mRight.forward();
for (int i = 0 ; i < 4 ; i++) { // Do FOUR times
  mLeft.forward();  // Go forwards (side i)
  Delay.msDelay(1000);
  mLeft.stop();     // Turn a corner
  Delay.msDelay(200);
}
mRight.stop();
```

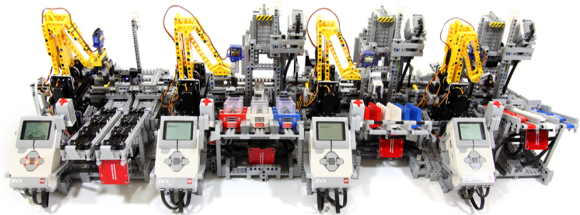
Creeper program

```
BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);

mL.forward(); // Start the ball rolling
mR.forward();

LCD.getInstance().drawString("Its all quiet", 2,1,0);
NXTSoundSensor us = new NXTSoundSensor(SensorPort.S1);
SampleProvider sp = us.getDBAMode();
float[] samples = new float[1];
sp.fetchSample(samples, 0); // Read Ahead Paradigm
while (samples[0] < 0.5) {
  // just keep swimming....
  sp.fetchSample(samples, 0); // Read Ahead for next loop.
}
LCD.getInstance().drawString("Wow - what a noise", 2,1,0);
mL.stop();
mR.stop();
```

Lego Does Lego



This is above and beyond

A Robot is...

Wiki

There is much discussion about which machines qualify as robots, a typical robot will have several, though not necessarily all of the following properties:

- Is not 'natural' i.e. has been artificially created.
- Can sense its environment.
- Can manipulate things in its environment.
- Has some degree of intelligence, or ability to make choices based on the environment, or automatic control.
- Is programmable.
- Can make dexterous coordinated movements.
- Appears to have intent or agency

Is this a robot?

Is this a robot?

Robot versus Remote Control

- Robot: has some control over itself
- Remote controlled machine: all decisions made by human operator

A Robot is...

The International Standard Organisation defines a *robot* (ISO 8373) as:

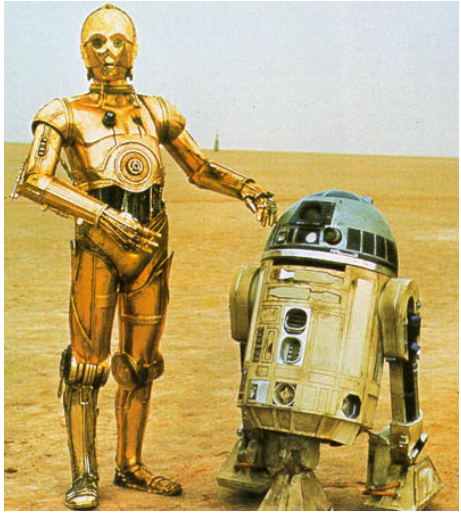
*An **automatically controlled**, **reprogrammable**, **multi-purpose**, **manipulator programmable** in three or more axes, which may be either fixed in place or mobile for use in industrial automation applications.*

But, how do we recognise a robot

Joseph Engelberger, a pioneer in industrial robotics, once remarked:

I can't define a robot, but I know one when I see one.

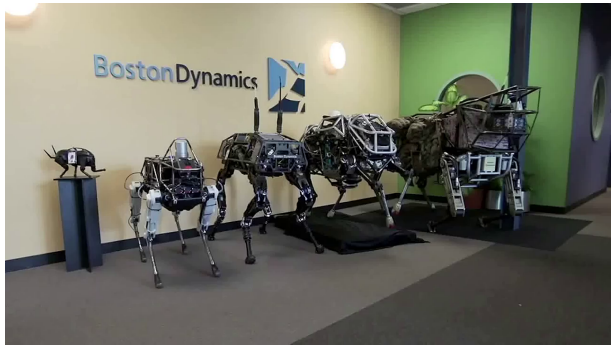
Robots in Sci-Fi, and beyond



Robots in Sci-Fi, and beyond

Robots in Sci-Fi, and beyond

Robots in Sci-Fi, and beyond



Just Wow - but is it a robot?

Are people Robots?

Are we Robots? Another Definition

- A robot is not 'natural' and appears to have intent or agency
- Can sense and manipulate things in its environment.
- Has some degree of intelligence: makes choices and/or runs autonomously.
- Is programmable.
- Can make dexterous coordinated movements.

Are people Robots?

Are we Robots? Another Definition

- A robot is not 'natural' and appears to have intent or agency
- Can sense and manipulate things in its environment.
- Has some degree of intelligence: makes choices and/or runs autonomously.
- Is programmable.
- Can make dexterous coordinated movements.

People aren't Robots

We aren't artificial.

We aren't very good at most tasks.

Senses and Sensors

A sense is a source of information about the external world: an input.

- You have all been taught to name the (five) human senses.
- What (extra) senses do other animals have?
- What senses can a robot have beyond these animal senses?

Standing up

How come, when you are standing, you don't (normally) fall over?

Senses and Sensors

A sense is a source of information about the external world: an input.

- You have all been taught to name the (five) human senses.
- What (extra) senses do other animals have?
- What senses can a robot have beyond these animal senses?

Standing up

How come, when you are standing, you don't (normally) fall over?

Hard senses to build into a robot - involving many sensors and much processing

Equilibrioception is balance or acceleration: Sensor in cavities in the inner ear.

Proprioception is body awareness: Sensors in every joint and muscle.

LEGO EV3™

LEGO Mindstorms EV3 is a programmable robotics kit released by LEGO in late August 2013.

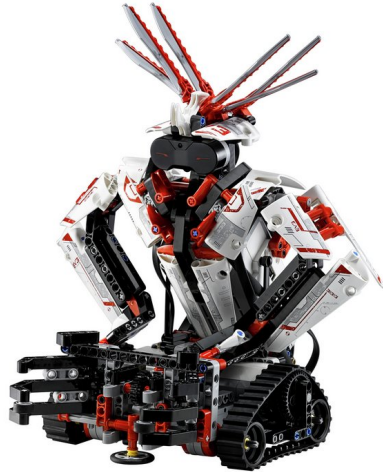


Figure 1: A walking LEGO robot

Why LEGO EV3

We study computer science not engineering

- The mechanical side is simple.
- It is (relatively) cheap, safe and flexible.
- We can program it using Java.
- We can build great robots.

The EV3 brick!

The brains of the outfit: the EV3 computer brick

- A 32-bit ARM9 300MHz main processor
- 256 KB flash memory (For the operating system)
- 64 MB RAM (For your program!)
- 178*128 pixel grayscale LCD (Nice, but small)
- A single USB 2.0 port and Bluetooth (Communicating, downloading programs etc)
- Four input ports (For four I2C, UART or Analogue sensors)
- Four output ports (For four Servo motors)
- A speaker (Play sound files at sampling rates up to 16 kHz)
- MicroSDHC slot (used for booting EV3Dev, and saving results)

What we give you to play with at RHUL in your robotics kit

Motors and Sensors

- Four (servo) motors with rotation sensors.
- *One touch sensor. Just a switch really!
- One colour sensor with a built in red LED.
- One NXT sound sensor.
- One ultrasonic sensor for distances.
- *One gyroscope sensor for rotational speed. .
- *Some extra Lego pieces.
- A Lego EV3 robot brain.
- A rechargeable battery charger
- Some Wires to connect Motors and Sensors

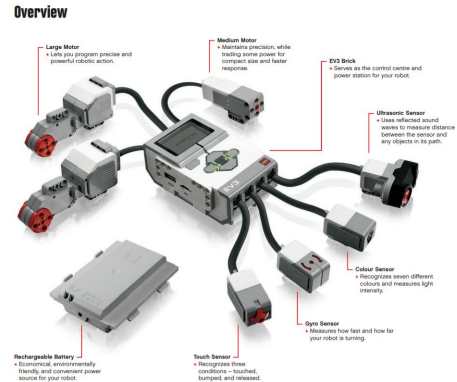


Figure 2: An EV3 brick and sensors

(*) We do not supply the gyroscope or touch sensors and Lego pieces until needed.

Getting Started

Aims of the Programming Lab Robotics Project

We teach Robot Programming because:

- It is fun to see programming in action.
- By building and programming robots students will see why:
  Accurate programming is important: It hard to debug a failing robot.
- By programming embedded systems students will see why:
  Advanced programming constructs are useful.: Robot programs are complicated.

Aims of the Programming Lab Robotics Project

We teach Robot Programming because:

- It is fun to see programming in action.
- By building and programming robots students will see why:
  Accurate programming is important: It hard to debug a failing robot.
- By programming embedded systems students will see why:
  Advanced programming constructs are useful.: Robot programs are complicated.

...plus

- Creativity — making cool stuff
- Problem Solving — this is a hands-off course

Objectives of the Programming Lab Robotics Project

At the end of this course students will:

- Know how to use the Eclipse IDE to write and compile Java programs.
- Know how to control LEGO EV3 robots using simple EV3Dev programs.
- Understand the use of loops, decisions and threads for robot control.

Course Requirements

Videos

- Watch some video lectures (fun and interesting) about Lego EV3 Robotics.
- Do five worksheets (**with deadlines**: assessed by TAs in group meetings).

Robotics Lab

- Do a group project on robotics.
- Write a final individual project report.
- Do a group presentation of your project.
- Make a Demo Video of your robot for the public.

Logistics of Group Work in COVID times

The Robot Holder is key to the group

- One group member needs to have the Basic Robbie robot.
- Robbie comes with a rechargeable battery and mains charger.
- This Robot Holder has certain responsibilities.
- The Robot Holder runs code from all the group members.
- ...and shows the group what is happening on zoom/teams/discord...
- You can change Robot Holder - but do it COVID safely.
- Keep Checking Moodle, Piazza and your email
- Keep the robot charged up.

Succeeding in Groups

Worksheets and Progress

- Worksheets help you to pace your progress (One+ per week).
- Every worksheet needs you to program the robot.
- You submit the code on Moodle.
- Stronger group members should run their own tutorials.
- Course leaders and/or tutors will be available at all sessions to help.
- Use the Piazza forum to ask questions
- We can help to fix robots that go bad.
- **Everyone codes.**

LEGO EV3™

LEGO Mindstorms EV3 is a programmable robotics kit released by LEGO in late August 2013.

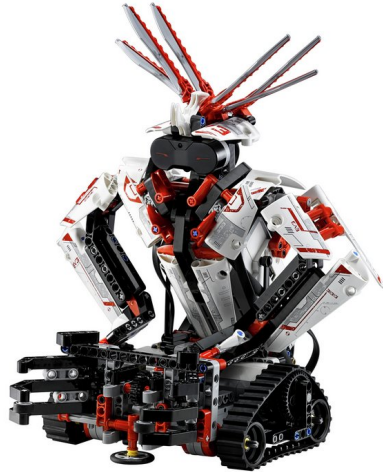


Figure 1: A walking LEGO robot

Why LEGO EV3

We study computer science not engineering

- The mechanical side is simple.
- It is (relatively) cheap, safe and flexible.
- We can program it using Java.
- We can build great robots.

The EV3 brick!

The brains of the outfit: the EV3 computer brick

- A 32-bit ARM9 300MHz main processor
- 256 KB flash memory (For the operating system)
- 64 MB RAM (For your program!)
- 178*128 pixel grayscale LCD (Nice, but small)
- A single USB 2.0 port and Bluetooth (Communicating, downloading programs etc)
- Four input ports (For four I2C, UART or Analogue sensors)
- Four output ports (For four Servo motors)
- A speaker (Play sound files at sampling rates up to 16 kHz)
- MicroSDHC slot (used for booting EV3Dev, and saving results)

What we give you to play with at RHUL in your robotics kit

Motors and Sensors

- Four (servo) motors with rotation sensors.
- *One touch sensor. Just a switch really!
- One colour sensor with a built in red LED.
- One NXT sound sensor.
- One ultrasonic sensor for distances.
- *One gyroscope sensor for rotational speed. .
- *Some extra Lego pieces.
- A Lego EV3 robot brain.
- A rechargeable battery charger
- Some Wires to connect Motors and Sensors

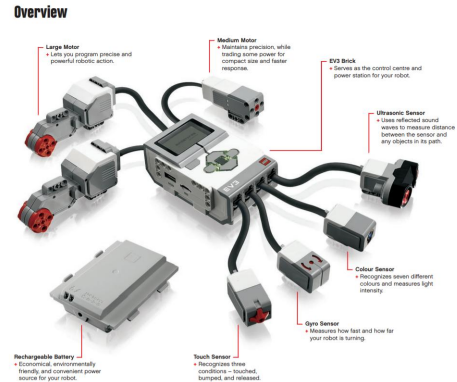


Figure 2: An EV3 brick and sensors

(*) We do not supply the gyroscope or touch sensors and Lego pieces until needed.

Projects

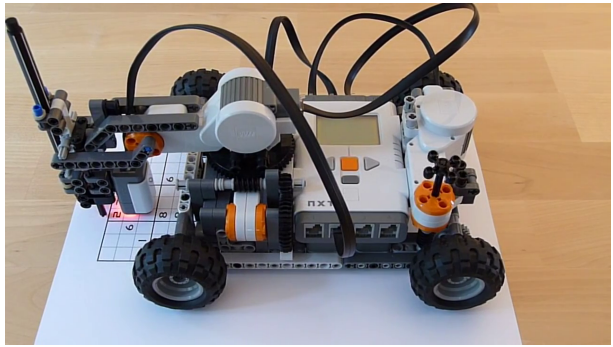
Kinds of Lego Robot

- Brilliant Table Toddler (maybe maps the table).
- Inspired Wall Follower (maybe maps the room).
- Clever Line Follower (maybe decides what to do at junctions).
- Sublime Object Counter (could sort objects as well).
- Maze solver, Card Player, Musical Instrument player, Intelligent Cars...
- Towers of Hanoi, ..., anything else agreed by me!

Moodle

There is a much more extensive list on Moodle.

A Lego robot that reads a Sodoku, Solves it and writes in the answer.



Basic questions to ask when you see a new robot

- How many effectors does this robot have (that change the world)?
- How many sensors does this robot have (that notice the world)?
- How could I improve this robot?

A Robot Design Exercise

Robot Design - Learning by doing - You will need a sheet of A4 paper

If you're watching a video - pause it now

Design a robot that tries to stay 30cm from the wall on its left.

- Draw a rectangular robot.
- Your robot should have distance sensors and motors.
- Show the positions, names (and directions) of the sensors and motors.

Write a short structured English description of the `main` method.

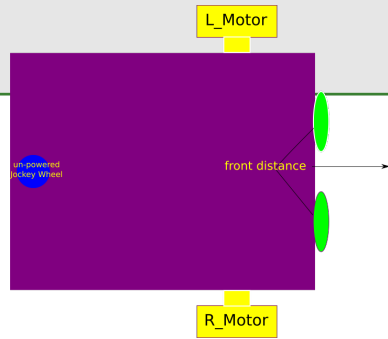
- Send messages to motors to make them turn.
- Send questions to the distance sensors to ask them their value.
- Use a WHILE loop, variables and IF statements.

Can your robot begin anywhere is the room and find a wall?

What does the robot do when it hits unexpected corners?

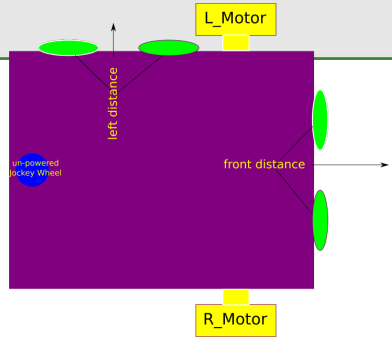
A (first) solution

```
TURN L_Motor ON
TURN R_Motor ON
DO FOREVER // An infinite while loop
  IF front_distance < 30 THEN
    TURN 90 degrees RIGHT // using L_Motor and R_Motor
  FI
OD
```



A (second) solution

```
WHILE battery level > 7v DO
  IF left distance < 25 TURN wheels RIGHT a bit.
  IF left_distance > 25 AND left_distance < 35 POINT STRAIGHT
  IF front_distance < 30 TURN RIGHT maximum.
  IF left_distance > 35 TURN wheels LEFT a bit.
ELIHW
```



A (nother) solution

```
WHILE battery level > 7v DO
  CALL KEEP_LEFT
  IF CALL AVOID CRASH says "its safe" THEN
    speed up a bit (bounded by MAX_SPEED)
  FI
ELIHW
```

```
PROC KEEP_LEFT
  Get left_distance AS left_one
  Wait until 1/10 sec passed.
  Get left_distance AS left_two
  Get angle from left_one and left_two
  Adjust turn to compensate.
```

```
PROC AVOID_CRASH // Return true if no wall seen
  IF front_distance < 100 THEN
    slow down a fair amount (but do not stop)
    IF front_distance < 30 THEN
      STOP
      WHILE front_distance < 80
        Turn right 5 degrees on spot.
      ELIHW
    FI
    return "not safe"
  FI
  return "its safe"
```

Working in a group

Lego Groups

Work starts this week!

- Do you know who is in your group?
- Do you know which session your group must attend?
- Have you decided when you will meet as a group **before** your session?
- Have you read the material on **Moodle**?

Moodle and Piazza and Social Media

Moodle page

- All lecture slides will be available (updated) on Moodle.
- Course Documentation will be available on Moodle.
- Coursework and Presentation Submission will be using Moodle.
- Worksheets will be available and submitted through Moodle.

The Piazza Forum

- Anonymous Questions can be asked on Piazza.
- Announcements will be made on Piazza.

WhatsApp™

Form a Social Media group. Have regular online meetings.

Java for Robotics

Twice

Wittgenstein

A serious and good philosophical work could be written consisting entirely of jokes

Twice

Wittgenstein

A serious and good philosophical work could be written consisting entirely of jokes

A relevant joke (Russian)

An engineer is digging a tunnel under a river.

He starts teams digging from both sides hoping to meet in the middle.

When he is asked what happens if they don't meet he replies:

Twice

Wittgenstein

A serious and good philosophical work could be written consisting entirely of jokes

A relevant joke (Russian)

An engineer is digging a tunnel under a river.

He starts teams digging from both sides hoping to meet in the middle.

When he is asked what happens if they don't meet he replies:

The Punchline

"Well, then we will have two tunnels"

Learning programming (again) for robotics

A Robot uses Event Driven Programming

- Small amounts of simple Java
- The focus is on doing interesting things as quickly as possible
- The challenge is to understand robot behaviour
- The other challenge is to learn the ideas behind event driven programming

Running your code on a Lego EV3 robot

There is a (specialised) Operating System and JVM inside your Brick

- Programs are run remotely from your PC using a special brickrun command
- In order for EV3Dev to run your code you have to Upload it to the brick using SCP.
- You (or the robot holder) always upload using the Maven Plugin.
- It is nicest to program your code using the Eclipse programming environment.
- Write English - Write Code - Check Code - Upload Code - Run Code - Fix Code.

Example: Holy Smoke

```
GraphicsLCD screen = LCD.getInstance();
screen.clear();
screen.drawString("Version 1.2", 0, 0, 0);
screen.drawString("HOLY", 0, 10, 0);
Delay.msDelay(1000);
screen.drawString(" SMOKE", 0, 20, 0);
```

Objects (like `LCD`) and Messages (like `drawString`)

- We are sending messages to a Java object called `screen`
- We are sending the message `drawString`
- Any additional information for the object we put into brackets.
- We are also sending a message to the `Delay` object.

Example: Sound it out

```
Sound speaker = Sound.getInstance();
speaker.beep();
speaker.twoBeeps();
speaker.playTone(2000, 1);
```

Surely you can guess what will happen when you run this code

Of course, you never need to guess: just upload the code and run it!

Motors are Individuals

Built in Objects - on the EV3 brick itself

Objects which are built in (like `Sound` and `LCD`) can just be used.

There are other builtin objects that we can work with. For example:

- `Button.ENTER`,
- `MotorPort.A`, `MotorPort.B`, `MotorPort.C` and `MotorPort.D`.

You have to plug motors into one of the motor ports (A,B,C or D)

- An NXT motor and an EV3 Large motor require different (electrical) signals.
- So, using motors is harder than using built in objects.
- We need to get the JVM to build motors, like an `EV3LargeRegulatedMotor`.

Example: Motorise this

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);

mLeft.setSpeed(720);// 2 Revolutions Per Second (RPS)
mRight.setSpeed(720);

mLeft.forward();
mRight.forward();
Delay.msDelay(1000);
mLeft.stop();
mRight.stop();

mLeft.close();  // Things need to be closed sometimes.
mRight.close();
```

Maybe you can work out what this program will do?

Again, if you cannot work it out, upload and run it. Computers do not break.

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project
- Type some programs into Java classes.

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project
- Type some programs into Java classes.
- Coorrect and improve two different programs for Robbie.

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project
- Type some programs into Java classes.
- Coorrect and improve two different programs for Robbie.
- Upload and run programs on Robbie!

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project
- Type some programs into Java classes.
- Coorrect and improve two different programs for Robbie.
- Upload and run programs on Robbie!
- Answer a few short questions and pass your first milestone.

Worksheet One

In this first week's lab work you will:

- Start Eclipse, create a Maven Project
- Type some programs into Java classes.
- Coorrect and improve two different programs for Robbie.
- Upload and run programs on Robbie!
- Answer a few short questions and pass your first milestone.
- Be kind and polite - read the Code of Practice.

Basic Lab Rules

Project Basics

- Work in your group - but divide up the tasks. The Project Groups are on Moodle.

Basic Lab Rules

Project Basics

- Work in your group - but divide up the tasks. The Project Groups are on Moodle.
- The Working Together document explains how to program together in a team.

Basic Lab Rules

Project Basics

- Work in your group - but divide up the tasks. The Project Groups are on Moodle.
- The Working Together document explains how to program together in a team.
- Java Robotics Project > Essential Reading is essential reading for robotics.

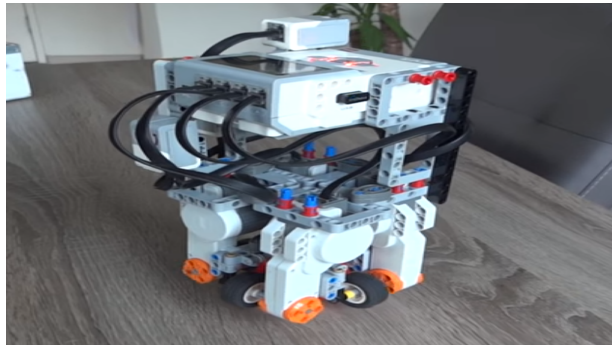
Basic Lab Rules

Project Basics

- Work in your group - but divide up the tasks. The Project Groups are on Moodle.
- The Working Together document explains how to program together in a team.
- Java Robotics Project > Essential Reading is essential reading for robotics.
- This essential reading is supported by some How-To videos.

Making Robots Behave
Solving Some general Robotics Problems

Ballbot - it balances on a ball - and moves in response to Bluetooth commands



Basic questions to ask when you see a new robot

- How many effectors does this robot have (that change the world)?
- How many sensors does this robot have (that notice the world)?
- How could I improve this robot?

Classical problems for a mobile robot

Where we are headed!

Problem 1: How do I get around (Navigation)

Many robots are mobile and have to navigate in the world

- They have Wheels, Legs, Ball-bearings or whatever Daleks use.
- They have a `Map` which is an internal representation of their world.
- They know their starting `Pose` - where they are and where they are facing
- They have a desired final `WayPoint` - which is where they want to be.
- They use an algorithm to find a `Path` through the `Map`.
- They must turn this `Path` into a series of motor commands.

Problem 2: The lost robot problem

Its not enough to plan a path. When I move I get lost!

- Every time a robot moves it may slip or the motors may be poorly calibrated.
- After the move the robot is less sure about where it is.
- It can use odometry to guess, using where it started and the attempted move.
- Using odometry also called dead reckoning was the cause of many shipwrecks.
- Robots need to cheat to keep on track.
- Try using dead reckoning yourself with a blindfold.

Finding where you are - using sensors and a supportive infrastructure

Use (many) sensors and build the robot environment to help it cheat

- Computer vision - look for or follow a known image.
- Distance Sensors - distance from the nearest object is always good to know.
- GPS - definitely cheating!
- Compass - you can get confidence about your direction - unless you are indoors.
- Rotation Sensors - you know how far wheels actually turned - but tyres still slip.
- Help me ...someone tells you where you are! - Guided robotics is fine.
- Put the robot on rails (like Docklands Light Railway)
- Give the robot lines to follow (like Amazon robots)
- Give the robot end stops with switches (so that it can re-register its position)
- Put the robot on a fixed base (like car building robots)

Modelling the world
Working with Java representations of Motors.

How do we begin to make this work? Modelling the World

We need to represent hardware and interact with the representation

- A class instance is often the program's *representation* of some real world object.
- A Java *object* `mLeft` represents a motor attached to LEGO EV3 port A.

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
```

- `mLeft` is not a motor.
- The code will compile and run even if you plug a lamp into Motor Port A.
- In robotics it is important to have objects that mediate between us and hardware.

Motor Objects - providing generalizable code

Question

What *state* and *methods* might we need for a `BaseRegulatedMotor` object?

Possible Answers

State Current actual speed, tachometer, desired speed, desired direction, complicated parameters...

Methods forward, setSpeed, setDirection, setAcceleration, isStalled ...

Its a bit Common

`BaseRegulatedMotor` is a class defining the common characteristics of a number of classes.

These methods can be used by the "real" motor classes such as `EV3LargeRegulatedMotor`.

By passing the base class around we could swap motor types without changing much code.

Even though the different motor's require different line voltages/signals.

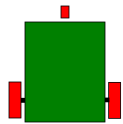
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.



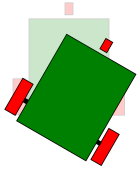
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.



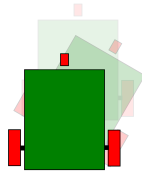
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.



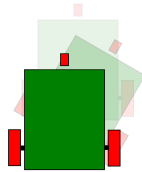
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.

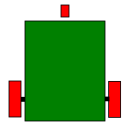


Going forwards

```
mLeft.rotate(360,true);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn and does not wait.

JVM starts the second motor (nearly) straight away.



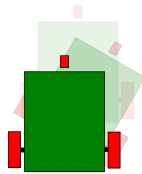
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.

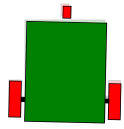


Going forwards

```
mLeft.rotate(360,true);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn and does not wait.

JVM starts the second motor (nearly) straight away.



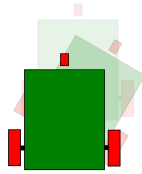
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.

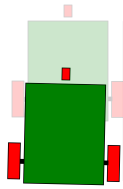


Going forwards

```
mLeft.rotate(360,true);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn and does not wait.

JVM starts the second motor (nearly) straight away.



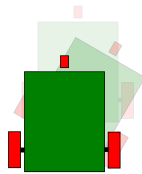
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.

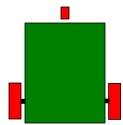


Going forwards

```
mLeft.rotate(360,true);
mRight.rotate(360,true);
```

JVM tells both `mLeft` and `mRight` to rotate without waiting.

for the other motor to finish.



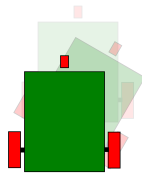
Non-blocking Calls and Synchronisation - making movement possible

Going forwards?

```
mLeft.rotate(360);
mRight.rotate(360);
```

JVM tells `mLeft` to rotate one turn.

JVM then waits until this has finished before rotating `mRight`.

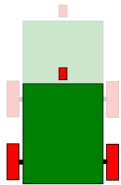


Going forwards

```
mLeft.rotate(360,true);
mRight.rotate(360,true);
```

JVM tells both `mLeft` and `mRight` to rotate without waiting.

for the other motor to finish.



Motors moving us along the path

We planned a path - we need to program it correctly

Two motors go forward for approximately one turn, at nearly the same time.

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);
mLeft.forward(); // start, mLeft replies immediately.
mRight.forward(); // start, mRight replies immediately.
Delay.msDelay(1000); // Delay waits one second before replying.
mLeft.stop();
mRight.stop();
```

Motors moving us along the path

We planned a path - we need to program it correctly

Two motors turn exactly 720° at nearly the same time.

```
// Shorter and better
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);
mLeft.rotate(720,true); // turn now, reply immediately.
mRight.rotate(720,false); // turn now, reply when finished.
```

It is very important to write good code.
Good code is readable and modifiable

Do it Again

Go round a square - Writing English first!

Obviously we can go round a square by doing the following eight steps:

1. Go Forward.
2. Turn Left
3. Go Forward.
4. Turn Left
5. Go Forward.
6. Turn Left
7. Go Forward.
8. Turn Left

Square Dancing Robot

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);

mRight.forward();
mLeft.forward();  Delay.msDelay(1000); // Both motors on for 1s - go forwards (side 1)
mLeft.stop(); Delay.msDelay(200);      // Stop mLeft - turn a corner for 0.2s
mLeft.forward(); Delay.msDelay(1000);  // Both motors on for 1s - go forwards (side 2)
mLeft.stop(); Delay.msDelay(200);      // Stop mLeft - turn a corner for 0.2s
mLeft.forward(); Delay.msDelay(1000);  // Both motors on for 1s - go forwards (side 3)
mLeft.stop(); Delay.msDelay(200);      // Stop mLeft - turn a corner for 0.2s
mLeft.forward(); Delay.msDelay(1000);  // Both motors on for 1s - go forwards (side 4)
mLeft.stop(); Delay.msDelay(200);      // Stop mLeft - turn a corner for 0.2s
mRight.stop();     // and stop.
```

Doesn't this code annoy you even though it works?

Cut and paste has no place in code - hard to read or modify or even test

For Square Dancing Robot

If you wrote it in English first you would always have it right!

Refactor the nasty cut and paste smell with a Loop.

1. Do FOUR times:
   - 1.1 Go Forward.
   - 1.2 Turn Left

```
BaseRegulatedMotor mLeft = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mRight = new EV3LargeRegulatedMotor(MotorPort.B);
mRight.forward();
for (int i = 0 ; i < 4 ; i++) { // Do FOUR times
  mLeft.forward(); Delay.msDelay(1000); // Go forwards (side i)
  mLeft.stop(); Delay.msDelay(200);     // Turn a corner
}
mRight.stop();
```

Getting More information

Javadoc and the EV3Dev API

An Applications Programmer Interface (API)

- An Applications Programmer Interface (**API**) is a set objects added to Java.
- An API is aimed at a particular domain: graphics, robotics etc.,.
- The EV3Dev API is described on a **Javadoc** web site. **Bookmark this site.**
  `https://ev3dev-lang-java.github.io/docs/api/latest/ev3dev-lang-java/index.html`
- The Javadoc describes `EV3largeRegulatedMotor` which extends `BaseRegulatedMotor`.
  Click through to find out the methods of a `BaseRegulatedMotor`.
- Also, a `BaseRegulatedMotor` implements the `RegulatedMotor` interface.
- This is why we can say that a `EV3LargeRegulatedMotor` is a `BaseRegulatedMotor` which is a `RegulatedMotor`.

Information Sources

Read all about it

- Moodle: **Essential Reading**, Lecture Slides (and videos), Worksheets.
- Tutorials: Many places on the web.
- Sample Code: `https://github.com/ev3dev-lang-java/examples`
- Source Code (of EV3Dev itself): `https://github.com/ev3dev-lang-java/ev3dev-lang-java`
- The EV3Dev API: `https://ev3dev-lang-java.github.io/docs/api/latest/ev3dev-lang-java/index.html`

Ask Questions

- At the Group Labs
- At the online Q+A session
- Any time (anonymously) on Piazza

Hints for Better Programming

Checking, Waiting and Sensors

Getting a Status Report

- A Lego motor is also a **sensor**.
- We can ask a `BaseRegulatedMotor` object `mL` how far it has turned: `mL.getTachoCount()`.
- It is always useful to show status on the screen: `LCD.getInstance().drawInt(mL.getTachoCount(),0,0,0);`

Getting an Object to Wait

- `Button.ENTER` represents the big grey ENTER button on the LEGO EV3.
- `Button.ENTER.waitForPressAndRelease()` returns after ENTER is tapped.
- It is **normal to wait** for the button tap at the start of your program.
- Display a **Version message** on a splash screen while waiting for the tap.

Worksheet two

You will:

- Write a program to make Robbie move forward until the ENTER button is pressed.
- Write a program to make Robbie move round a one meter square.
- Experiment with other `BaseRegulatedMotor` methods.
- Answer a few short questions and pass the second milestone.

Your Project

It is never too soon to organise and plan

- Define clear group roles - but everyone programs
- Brainstorm project ideas...
- Choose your project and begin designing the hardware now.
- Everyone programs. Fail the course by not programming
- If lockdown continues then we may have to change the project specification.

Robot Senses

Inputs

Programs require input

Without input a program is boring. It can only answer one question.

Inputs

Programs require input

Without input a program is boring. It can only answer one question.

Robots require input

Without input a robot is an **automaton**. It always does the same thing.

Inputs of various kinds

Built in sensors

- Buttons on the brick (the brick has six buttons you can watch).
- Tachometer (rotation sensor) in a Lego motor
- Battery Voltage (useful to gracefully fail)

External sensors - these need connecting to the robot

- Touch sensor (just another switch) (wire connection to Ports 1-4)
- Ultrasound (Distance) sensor (wire connection to Ports 1-4)
- Sound Sensor (very hard to program with) (wire connection to Ports 1-4)
- Colour Sensor (actually also a torch) (wire connection to Ports 1-4)
- Gyro Sensor (how fast am I spinning?) (wire connection to Ports 1-4)
- Use an Android phone to do Colour Tracking or QR code recognition (Bluetooth)

Motors, LCD screens etc., are easy

Objects that represent robot hardware

The screen to write on is an abstraction represented by `LCD`

- Sending a message to `LCD` changes the physical object that it represents.
- There is only one screen so we do not need to create an `LCD` object for it.
- Any `drawString` must be for the only screen.
- All methods of the `LCD` class operate on the same `LCD` screen.
- We just say `LCD.getInstance().drawString('Hello world', 0, 10, 0)`.

Woking (again) with Motors

Motor ports may be unused, or could have any kind of motor plugged in

- We cannot say `BaseRegulatedMotor.forward()` - not all motors are the same.
- All motors can do whatever a `BaseRegulatedMotor` can do - like go forwards.
- There may be four `BaseRegulatedMotor` objects in any EV3Dev program.
- These are just the Java abstractions for motors attached to motor ports.
- You use `BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A)`
  - It would be sensible to have an EV3 Large Motor stuck into Port A on your brick.
  - You can send a `forward` message to the Java object `mL`
  - A *real* motor attached to port A will begin to turn forward!
  - If it is not an EV3 large motor on port A then anything could happen (or nothing)

Four EV3 Motor ports

Creating Java Motors

- We have asked the JVM before to create a `BaseRegulatedMotor` object.
- We used `BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A)`
- We send this Java object a message. It then accesses Port A for us.
- Actually it sends a message to the java object `MotorPort.A`
- Presumably the `MotorPort.A` Java object (Somehow) sets voltages on wires.
- We assume that there is a large EV3 motor plugged into Port A.
- You can plug something else into this port.
- The voltages produced are correct for an EV3 large motor.

Java objects and real things

Fixed parts

Motor and Sensor ports are the same for any robot built on a EV3 brick.

- The EV3 brick has four motor ports.
- The EV3 brick has four sensor ports.

So they are constructed by the JVM in the `MotorPort` and `SensorPort` classes.

Mediation and the JVM

You use objects to let JVM mediate between you and *real* robot parts.

Without being able to use the method `forward` from the `BaseRegulatedMotor` class you would need to know how precisely to adjust the voltages (and timings) on the wires in Motor port A to make the motor attached to this port drive forward.

This is called Bit Banging - and is hard to get right.

Motors, LCD screens etc., are easy

Sensors are a bit harder

Sensor objects and real sensors

Sensor types

- You can plug anything into a sensor port (or a motor port) on the EV3 brick.
- You have to interpret the readings (voltages).
- You also have to decide how often to check the readings.

Feeling Bright

- There cannot be a Base Sensor that all sensors look like.
- An `EV3ColorSensor` would have different methods from an `EV3TouchSensor`.
- Both objects have mechanisms to fetch the "value" of the sensor.
- In both cases the Java object knows how to *interpret* the sensor voltages.

`SensorPort` and `Sensor` objects

Raw values

- You could read the raw value of any sensor port.
- Its not clear what you would do with such a value.
- Best let a `Sensor` object, like an `EV3TouchSensor` deal with it.
- You create a `EV3TouchSensor` object which is looking at a sensor port.
- It is sensible to have an actual touch sensor plugged into that port.

EV3 Sensors are very different from each other

Sensors do not even all return the same number of values!

- `EV3TouchSensor` can tell you a `boolean`.
- `EV3ColorSensor` can tell you three `int` primary colour values.
- `NXTSoundSensor` can tell you a `float` sound level (Decibels).
- `EV3GyroSensor` can tell you a `float` speed of rotation (degrees/second).
- `EV3UltrasonicSensor` can tell you a `float` distance (cm).

Using an EV3 colour sensor

There are four sensor ports: 1, 2, 3 and 4.

The colour sensor can be attached to any sensor port.

There are four motor ports: A,B,C and D

Port A is always attached to a motor if its attached to anything.

Writing to `SensorPort.S1`

- Sensor port 1 could have any sensor attached.
- If we plug a colour sensor into sensor port 1 then...
- ...we need to create a `EV3ColorSensor` object *attached* to `SensorPort.S1`.

All Sensors provide Samples

Sampling a colour sensor

Sequence of events

1. Create an `EV3ColorSensor` object to read a real colour sensor attached to Port 1.
2. Set the **colour sensor mode** by asking for the appropriate **sample** provider.
3. Ask the **sample** provider object for the light levels **sample**: `fetchSample`.
4. The **sample** provider object asks the hardware attached to Port 1 for a **sample**.
5. The **sample** provider replies to the message with its **interpreted value(s)**.

Anything (even a motor) might be plugged into Port 1.

Sample providers need to return an array of values

Colour sensors return three values (red, green, blue) in a sample

- A `SampleProvider` has the `fetchSample` method.
- We pass an array to the `fetchSample` method.
- The `SampleProvider` queries the actual sensor.
- The `SampleProvider` fills in the array.
- The sample provider can return more than one value.

Creating a colour sensor

Four sensor ports

There are four sensor port objects called:

- `SensorPort.S1;`
- `SensorPort.S2;`
- `SensorPort.S3;`
- `SensorPort.S4;`

Create Java objects to check sensor port 1 for light levels

```
float[] samples = new float[3]; // Create an array of three floats
EV3ColorSensor cs = new EV3ColorSensor(SensortPort.S1);
SampleProvider sp = cs.getColorMode();
sp.fetchSample(samples, 0); // Fill the array with colour levels
```

Samples are Lists

```
float[] samples = new float[3]; // create an array of three floats
EV3ColorSensor cs = new EV3ColorSensor(SensortPort.S1);
SampleProvider sp = cs.getColorMode(); // used to query the sensor

sp.fetchSample(samples, 0); // Fill my array with samples: [R,G,B]

LCD.getInstance().drawString("Blue Level: " + samples[2], 0, 0, 0);
// index runs 0..2
```

The modes of a sensor and the meaning of a sample

The mode describes the sample. A colour sensor has several modes.

- In one mode a sample has three values: the three primary colour values
- In another mode a sample has just one value: the ambient white light level.
- We set the sensor mode by asking for an appropriate `SampleProvider`.

```
EV3ColorSensor cs = new EV3ColorSensor(SensorPort.S2);
SampleProvider sp = cs.getAmbientMode();
float[] samples = new float[4]; // NOTE THE FOUR!
sp.fetchSample(samples, 0); // Now samples[0] contains the ambient light level.

sp = cs.getRGBMode();
sp.fetchSample(samples, 1);
// Now samples[1], samples[2], samples[3] contain the Red, Green and Blue levels.
```

Sensors need to be Calibrated and Normalised

Calibration of sensors

Calibration to make numbers correct

A thermometer (temperature sensor) needs to be calibrate to degrees Celsius.

- Stick it in ice water – mark it 0.
- Stick it in boiling water – mark it 100.
- Divide the distance by 100 and scribe on tick marks.

Robot engineers sometimes calibrate sensors to emphasise differences

We can calibrate (normalise) sensors to make small variations into big differences.

Calibrating a EV3 colour sensor

Normalising Samples

- We could just use an **ambient light sample provider**
- If we only ever get values between 0.45f and 0.47f then we have **a problem**.
- So we **normalise ambient light levels** and we will see several ways to do this.
- You normalise so that 0.0f and 1.0f represent the darkest and lightest levels.

Light levels

- Ambient light varies from day to day
- ...and from hour to hour ...
- Robots that depend on seeing lines and colours need to use normalised values.

Using a colour sensor - normalising ambient light levels

```
GraphicsLCD screen = LCD.getInstance();
EV3ColorSensor ls = new EV3ColorSensor(SensorPort.S1);
SampleProvider sp = ls.getAmbientMode();
float[] samples = new float[3];

screen.drawString("Darkest dark?", 2, 3, 0); Button.ENTER.waitForPressAndRelease();
sp.fetchSample(samples, 0);

screen.drawString("Point Somewhere", 2, 4, 0); Button.ENTER.waitForPressAndRelease();
sp.fetchSample(samples, 1);

screen.drawString("Brightest light?", 2, 2, 0); Button.ENTER.waitForPressAndRelease();
sp.fetchSample(samples, 2);

screen.drawString("Lowest = " + samples[0],2, 7, 0);
screen.drawString("Light value = " + samples[1], 2, 8, 0);
screen.drawString("Highest = " + samples[2], 2, 6, 0);

float normalised = (samples[1] - samples[0]) / (samples[2] - samples[0]);
screen.drawString("Normalised value = " + normalised, 2, 10, 0);
```

Program well - I hate magic numbers

```
final int HIGH = 2; final int NORMAL = 1; final int LOW = 0 ; final int PRINT_ROW = 10;

EV3ColorSensor ls = new EV3ColorSensor(SensorPort.S1);
SampleProvider sp = ls.getAmbientMode();
float[] samples = new float[3];

screen.drawString("Darkest dark?", 2, 3, 0); Button.ENTER.waitForPressAndRelease();
sp.fetchSample(samples, LOW);
screen.drawString("Point Somewhere", 2, 4, 0); Button.ENTER.waitForPressAndRelease();
sp.fetchSample(samples, NORMAL);
screen.drawString("Brightest light?", 2, 2, 0); Button.ENTER.waitForPressAndRelease();
screen.fetchSample(samples, HIGH);

LCD.drawString("Lowest = " + samples[LOW], 2, 7, 0);
LCD.drawString("Light value = " + samples[NORMAL], 2, 8, 0);
LCD.drawString("Highest = " + samples[HIGH], 2, 6, 0);

float normalised = (samples[NORMAL] - samples[LOW]) / (samples[HIGH] - samples[LOW]);
LCD.drawString("Normalised value = " + normalised, 2, PRINT_ROW, 0);
```

Looping until a change occurs.
Event driven programming (at last).

Loops

Waiting for something to change

- You want to do something until you notice a change.
- Say you just want to go straight ahead until you hear a loud noise.
- To do something over and over again you use a `while` loop.
- You can then say: *go forwards until you hear a loud noise.*

```
GO FORWARDS
WHILE ITS QUIET DO:
  PRINT "Its all quiet" ON THE LCD SCREEN
ELIHW
PRINT "Wow - what a noise" ON THE LCD SCREEN
```

Creeper program

```
BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);

mL.forward(); // Start the ball rolling
mR.forward();

NXTSoundSensor us = new NXTSoundSensor(SensorPort.S1);
SampleProvider sp = us.getDBAMode();
float[] samples = new float[1];
LCD.getInstance().drawString("Its all quiet", 2, 1, 0);
sp.fetchSample(samples, 0); // Read Ahead Paradigm
while (samples[0] < 0.5) {
  // just keep swimming....
  sp.fetchSample(samples, 0); // Read Ahead for next loop.
}
LCD.getInstance().drawString("Wow - what a noise", 2, 1, 0);
mL.stop();
mR.stop();
```

A light meter

Keeping Accounts in a variable

- You want to record the highest light value that your robot sees while it is running.
- You create a variable for putting numbers into: `float highestLightValue;`
- Now you need to update the value `if` you see a brighter light.
- Robot programs (like this one) run forever, or until a button is pressed!

```
STORE 0 IN highestLightValue
WHILE True DO
  IF sensorValue BIGGER THAN highestLightValue THEN
    STORE sensorValue IN highestLightValue
  FI
ELIHW
PRINT highestLightValue ON THE LCD SCREEN
```

Highlight program

```
GraphicsLCD screen = LCD.getInstance();
float highestLightValue = 0f;
EV3ColorSensor cs = new EV3ColorSensor(SensorPort.S1);
SampleProvider sp = cs.getRedMode();
float[] samples = new float[1];
while (true) {
  sp.fetchSample(samples, 0);
  if (samples[0] > highestLightValue) {
    highestLightValue = samples[0];
    screen.clear(); // Why is this needed?
    screen.drawString("Highest Light: " + highestLightValue, 1, 1, 0);
  }
}
```

Nervous program - its like a small dog!

```
BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);
EV3UltrasonicSensor us = new EV3UltrasonicSensor(SensorPort.S1);
SampleProvider sp = us.getDistanceMode();
float[] samples = new float[1];
while (true) {
  sp.fetchSample(samples, 0);
  if (samples[0] < 0.40f) { // 40cm
    mL.backward();
    mR.backward();
  }
  if (samples[0] > 0.60f) {
    mL.forward();
    mR.forward();
  }
  if (samples[0] < 0.60f && samples[0] > 0.40f) {
    mL.stop(); mR.stop();
  }
}
```

Using a `SampleProvider` to filter sensor values

Sensors, values, samples and normalisation

Inconvenience

You could read the raw values from the sensor port and convert them yourself.

Convenience

- `EV3ColorSensor` is just a convenience class.
- It wraps up a `SensorPort` and mediates the results
- We access the sensor using a `SampleProvider` that fills in an array of values.
- We can build normalisation/filtering into this object to make things (even) easier.
- Worksheet three: you use a `SampleProvider` to filter sensor values.
- We can use a normalising `SampleProvider`. I have given you one on GitHub.

The Best Way to calibrate (normalise)

We need to mediate

- We create an object to mediate between you and a vanilla sample provider.
- This object will remember the highest and lowest values.
- It will scale the values in between when you ask for a sample.
- We have written a class called **MinMaxFilter**. This is how we use one:

```
GraphicsLCD screen = LCD.getInstance();
EV3ColorSensor cs = new EV3ColorSensor(SensortPort.S1);
SampleProvider minmax_sp = new MinMaxFilter(cs.getAmbientMode());
screen.drawString("Move the robot around to calibrate light levels", 2, 8, 0);
screen.drawString("Press ENTER when you are done", 2, 9, 0);
minmax_sp.calibrate();
```

Now we can use **minmax_sp.fetchSample** to return (light) values from 0 to 1.

- This is particularly useful for sound and ambient light sensors.

A Calibrating Sample Filter

```
// Mediates between a sensor sample provider and the user to provide normalised values between the Min and Max values
// seen during a calibration phase.  Only works with sample providers returning a single value.
public class MinMaxFilter extends SampleProvider {
  private SampleProvider sp;   private float range = 1.0f;  private float offset = 0.0f;
  MaxMinColorFIlter(SampleProvider _sp) { // The actual sensor object will be used to get un-normalised values.
    sp = _sp;
  }
  public void calibrate() {  // This function is called once and then it keeps getting values until ENTER is pressed
    float samples = new float[1];
    float highValue = 0.0f;
    float LowValue = 1.0f
    while (!Button.ENTER.isPressed()) {
      sp.fetchSample(samples, 0);
      highValue = Math.max(highValue, samples[0]);
      lowValue = Math.min(lowValue, samples[0]);
    }
    offset = low_value;
    range = highValue - lowValue; // Before returning it stores the range and offset values for future use.
  }
  public int sampleSize() {
    return 1;
  }
  public void fetchSample(float[] sample, int index) {
    sp.fetchSample(sample, index); // delegate the fetchSample call to the actual sensor
    sample[index] = (sample[index] - offset) / range; // Adjust the result and put it into the returned array
    return;
  }
}
```

Worksheet Three

You will:

- Write Java programs to calibrate sensors.
- Write Java programs to:
  - follow a black line on the floor.
  - go forward until 50cm from a wall and then stop.
  - look intelligent:
    1. Do nothing until you clap your hands, then start moving forward.
    2. When you clap again turn right 90 degrees.
    3. When within 50cm of a wall stop.
- Answer a few short questions.

Threads and Navigation

Multi-tasking

JVM can multi-task

In fact JVM can do two things at once!

Just like running two programs on the same robot.

She can be simultaneously (nearly)

- listening to a sensor port in one task,
- while running your `main` program in another task.

She does this by regularly switching her attention between tasks.

Using a Thread to Watch a sensor

A Watching thread

Creating threads

- When EV3Dev begins, the JVM has just one task: your `main` method.
- Tasks are called threads in Java.
- You can create a new thread just to watch a sensor.
- The second thread could watch for a noise.
- The watcher thread can stop the motors (straight away).
- ...even if your main thread is waiting for a motor to finish turning.

Stopping in time

There is a problem while your robot is nicely going around a square

- Someone sticks their hand in the way of the robot
- The robot hits the hand and falls over
- Even though it has a touch sensor on the front

Ways to go forward - while watching - proper event driven

- POOR: Rotate the motors and just hope no-one gets killed!
- OKAY: Divide up the rotation into tiny little bits.
  Rotate one little bit at at time.
  Check the touch sensor before each little rotation.
- SAFE: Use a single `rotate` call.
  Start a new `Thread` to watch a sensor during this movement.

POOR: Not Stopping

```
public class TestWait {
  final static int TOTAL = 1440; // total number of degrees to turn the wheels

  public static void main(String[] args) {
    BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
    EV3TouchSensor ts = new EV3TouchSensor(SensorPort.S2);
    SampleProvider sp = ts.getTouchMode();
    float[] samples = new float[1];

    mL.rotate(TOTAL);
    LCD.getInstance().drawString("Finished", 2, 4, 0);
    Button.ENTER.waitForPressAndRelease();
  }
}
```

If the sensor is touched during the movement you do not notice.

OKAY: Polling the Sensor

```
public class TestWait {
  final static int TOTAL = 1440; final static int EACH_MOVE = TOTAL/100;
  public static void main(String[] args) {
    BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
    EV3TouchSensor ts = new EV3TouchSensor(SensorPort.S2);
    SampleProvider sp = ts.getTouchMode();
    float[] samples = new float[1];

    int remaining = TOTAL;  // remaining is how far we have left to turn the motors.
    while (remaining > EACH_MOVE) { // We have not yet finished - so turn some more.
      mL.rotate(EACH_MOVE);
      remaining -= EACH_MOVE;  // Deduct this rotation from the amount remaining to do.
      if (ts.isPressed()) break; // If, after this small turn, we notice an obstacle...
    }
    LCD.getInstance().drawString("Finished " + TOTAL, 2, 4, 0);
    Button.ENTER.waitForPressAndRelease();
}}
```

If the sensor is touched you might not notice.

SAFE: Watching the Sensor

```
public class TestWaitStop { // Now no-one gets killed!
  final static int TOTAL = 1440;
  public static void main(String[] args) {
    BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
    Thread watcher = new WatcherThread(mL);
    watcher.setDaemon(true); watcher.start(); mL.rotate(TOTAL);
    LCD.getInstance().drawString("Finished " + mL.getTachoCount(),2,4,0);
    Button.ENTER.waitForPressAndRelease();
}}
public class WatcherThread extends Thread {
  private BaseRegulatedMotor m;
  public WatcherThread(BaseRegulatedMotor _m) { m = _m; }
  public void run() {
    EV3TouchSensor ts = new EV3TouchSensor(SensorPort.S2);
    SampleProvider sp = ts.getTouchMode();
    float[] samples = new float[1];
    while (true) {
      sp.fetchSample(samples, 0);
      m.setSpeed(samples[0] > 0.5 ? 1 : 720);
      Thread.yield(); // Polite nod....
}}}
```

Using a Pilot to drive a vehicle

Two Wheeled Vehicles - some simple geometry

From Wheels and Axles to Straight Lines and Circles

Knowing the axle length and wheel diameter you can calculate motor rotate angles:

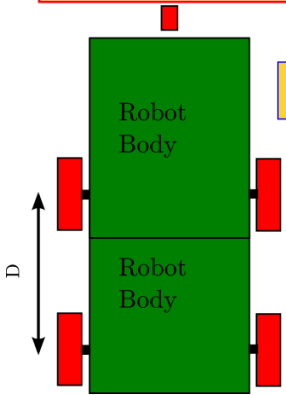
- to go a given distance
- to turn through a given angle.

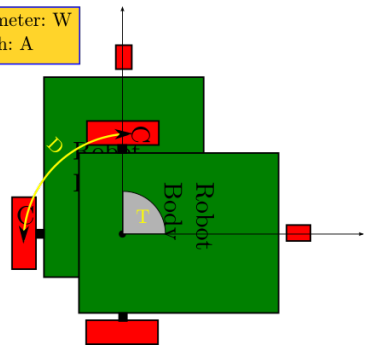
Two Wheeled Vehicles - some simple geometry

Distance Travelled: D
Wheel Rotations: R
Wheel Circumference: π * W
So R * π * W = D
R = D / (π * W)

Rotation Angle: T
Wheel C distance moved: D
Wheel C rotations: R
So R = D / (π * W)
But D / (π * A) = T / 360
So D = (π * A * T) / 360
Hence R = (A * T) / (360 * W)

Wheel Diameter: W
Axle Length: A





Pilots are brilliant

Pilots do the maths and run the threads - use them to mediate

- A `MovePilot` object controls two motors at once.
- Internally it creates threads which watch the motors.
- So it can start two motors rotating at the same time!
- Easier and better than you synchronising them.

Cars and Pilots

- Of course, a `MovePilot` does not correspond to a real world object.
- It mediates between your program and a pair of wheels.
- It provides simple methods like `travel`, `rotate` and `arc`.
- The `MovePilot` (differential) needs to know the wheel positions.
- These positions are stored in a `WheeledChassis` passed to the `MovePilot`.

Pilot Square

```
public class SquarePilot {
 public static void main(String [] args) {
  // Wheel Diameter 60mm, both wheels set 29mm from car centre
  BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
  Wheel wL = WheeledChassis.modelWheel(mL, 60).offset(-29);

  BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);
  Wheel wR = WheeledChassis.modelWheel(mR, 60).offset(29);
  Wheel[] wheels = new Wheel[] {wR, wL};
  Chassis chassis = new WheeledChassis(wheels, WheeledChassis.TYPE DIFFERENTIAL);
  MovePilot pilot = new MovePilot(chassis);
  for (int side = 0 ; side < 4 ; side++) {
    pilot.travel(300);
    pilot.rotate(90);
  }
 }
}
```

A Pose Provider

Odometry

- A `PoseProvider` is an object that tells you the current pose of the robot.
- It could:
  - Watch the `Pilot` and update its position when a move finishes (Odometry).
  - Ignore the `Pilot` and use GPS,
  - Use odometry and also update when it sees Landmarks using a Camera,
  - Use the distances from the walls, and a map, to work out where it is,
  - Use a BlueTooth connection to a human to tell it where it is.
- It could even combine a collection of methods to get the most reliable pose.
- It is quite tempting to write your own `PoseProvider`

The Odometry Pose Provider

Some EV3Dev Objects useful for navigation

- We use a `Pilot` to control the motors.
- The `Pilot` that we use for two driven motors is a `MovePilot`.
- If you have some other movement system then you may need to write your own `Pilot`.
- We also use a `PoseProvider` to find out where we are.
- The simple `OdometryPoseProvider` pose provider (in leJOS) uses odometry.
- It is tempting to extend the `OdometryPoseProvider` class to make it better.

Odometry Example

```
public class WhereAmI {
 public static void main(String[] args) throws Exception {
  // Wheel Diameter 60mm, both wheels set 29mm from car centre
  BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
  Wheel wL = WheeledChassis.modelWheel(mL, 60).offset(-29);
  BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);
  Wheel wR = WheeledChassis.modelWheel(mR, 60).offset(29);
  Wheel[] ws = new Wheel[] {wR, wL};
  Chassis chassis = new WheeledChassis(wheels, WheeledChassis.TYPE DIFFERENTIAL);
  MovePilot pilot = new MovePilot(chassis);
  PoseProvider poseP = new OdometryPoseProvider(pilot);
  for (int side = 0 ; side < 4 ; side++) {
   pilot.travel(300); pilot.rotate(90);
  }
  LCD.drawString(0,0,poseP.getPose().toString());
  Button.Enter.waitForPressAndRelease();
 }
}
```

Tips and Tricks

Posing Problems - robots need to cheat

- Do not rely on odometry
- Use touch sensors at the end of tracks
- Use marks on a track and a light sensor
- Use gears (fixed in a cage) and a geared track
- Use accurate distance sensors

A Robot with a Pilot and a watching Thread

You will be programming this behaviour

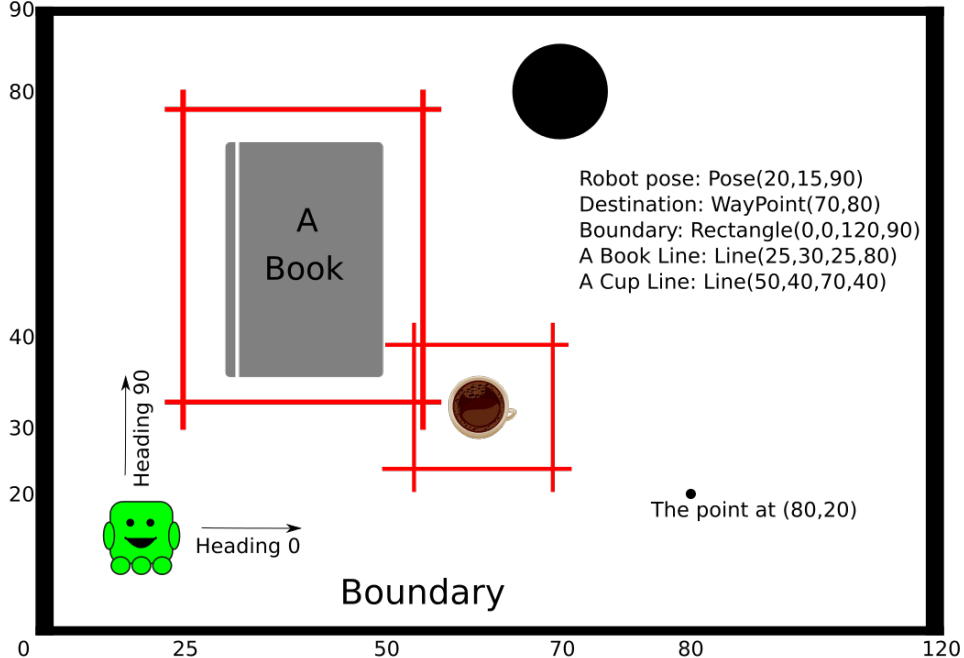
Except that this robot has rack and pinion (not differential) steering .

Navigation in IeJOS
Waypoints, Poses, Maps and Paths.

Finding your way around the leJOS way

Poses, Waypoints and Paths

- The leJOS world is defined by coordinates.
- Directions are relative to the x-axis.
- Rotation anti-clockwise increases the angle, so the y-axis is 90°.
- A `Pose` is the position and orientation of the robot.
- When a robot program begins its pose is $\langle (0, 0), 0^\circ \rangle$.
- A `Waypoint` is a position on the map.
- A `Path` is a list of `Waypoint` objects.


90
80
40
30
20
0
25
50
70
80
120
A
Book
Robot pose: Pose(20,15,90)
Destination: WayPoint(70,80)
Boundary: Rectangle(0,0,120,90)
A Book Line: Line(25,30,25,80)
A Cup Line: Line(50,40,70,40)
Heading 90
Heading 0
The point at (80,20)
Boundary

The Navigator

Some more leJOS Objects for navigation

- We use a `Navigator` to navigate a path.
- It sends a `Pilot` commands to move along a `Path`
- It also checks after at each `Waypoint` that it is on course using a `PoseProvider`.
- There is a simple `Navigator` class in leJOS that we can use.
- The `Navigator` constructor takes a `PoseProvider` object and a `Pilot`.
- Call `followPath` in your `Navigator` to begin the path.

Example Navigation using a set of known waypoints

```
public class PathFollowing {
 public static void main(String[] args) throws Exception {
  BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(MotorPort.A);
  Wheel wL = WheeledChassis.modelWheel(mL, 60).offset(-29);
  BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(MotorPort.D);
  Wheel wR = WheeledChassis.modelWheel(mR, 60).offset(29);
  Wheel[] wheels = new Wheel[] {wR, wL};
  Chassis chassis = new WheeledChassis(wheels, WheeledChassis.TYPE DIFFERENTIAL);
  MovePilot pilot = new MovePilot(chassis);

  PoseProvider poseP = new OdometryPoseProvider(pilot);
  Navigator navigator = new Navigator(pilot, poseP);
  Path route = new Path();
  route.add(new Waypoint(100,0)); route.add(new Waypoint(100,100));
  route.add(new Waypoint(0,100)); route.add(new Waypoint(0,0));
  navigator.followPath(route); // followPath returns immediately, so
  navigator.waitForStop(); // we wait for the Navigator to stop!
 }
}
```

Stopping a path to do something else

We can stop a `Navigator` in another `Thread` and resume it later.

- We can use a `Thread` to watch a sensor.
- Then we can stop a `Navigator` in the watching `Thread`.
- In the main thread we can wait until its safe to resume following the `Path`.

Example Navigation

```
public class PathFinding {
  public static void main(String[] args) throws Exception {
    float[] samples = new float[1]; // Updated with the distances.
    // Create Navigator (nav), Path (route) as usual.
    .....
    Thread t = new Watcher(nav, samples); t.start();
    nav.followPath(route); nav.waitForStop(); // The read ahead paradigm again
    while (!nav.pathCompleted()) {
      while (samples[0] < 0.25f) { // Start again when obstacle > 25cm
        Sound.beep(); Delay.msDelay(500);
      }
      nav.followPath(); nav.waitForStop();
    }
  }
}
public class Watcher extends Thread {
  private Navigator nav; private float[] dist;
  public Watcher(Navigator _nav, float[] _s) { nav = _nav; dist = _s;}
  public void run() {
    EV3UltrasonicSensor us = new EV3UltraSonicSensor(SensorPort.S1);
    SampleProvider sp = us.getDistanceMode();
    while (true) {
      sp.fetchSample(dist, 0);  // Fetch the distance into the shared samples array every time around this loop.
      if (dist[0] < 0.10f && nav.isMoving()) { nav.stop(); } // Obstacle!
    }
  }
}
```

Maps

A Map in leJOS is a set of lines - used to keep a robot away from danger

- We use a `LineMap` to tell the robot where things are in its environment.
- A `LineMap` object has a list of lines and a rectangular bounding box.
- We use the `Line` objects to mark out obstacles.
- We use a `PathFinder` to build a `Path` through a `LineMap`
- The `PathFinder` will not go outside the boundary, nor cross any `Line`.
- We tell the `PathFinder` the starting `Pose` and an ending `Waypoint`.
- The `ShortestPathFinder` uses a clever algorithm to find the shortest path.
- **A `PathFinder` will never cross a line in the map.**
- **A `PathFinder` is happy to go through the end of a line.**

Getting there quickly

```
public class PathFinding {
 public static void main(String[] args) throws Exception {
  // Create Navigator (nav) as usual.
  .....

  Line [] lines = new Line[4];
  lines [0] = new Line(-20f, 20f, 100f, 20f);
  lines [1] = new Line(-20f, 40f, 20f, 40f);
  lines [2] = new Line(-20f, 60f, 20f, 60f);
  lines [3] = new Line(-20f, 80f, 20f, 80f);
  Rectangle bounds = new Rectangle(-50, -50, 250, 250);
  LineMap myMap = new LineMap(lines, bounds);
  PathFinder pf = new ShortestPathFinder(myMap);
  Path route = pf.findRoute(new Pose(0,0,0), new Waypoint(0, 100));
  nav.followPath(route);
  nav.waitForStop();
 }
}
```

Behaviours, Project Extras and Grading

First - more about good programming!

Feedback

```
int x; boolean t; x = 0; // WTF
for (x = 1; x <= 5; x = x + 1){mLeft.forward();
Delay.msDelay(173);} // WTF (three or four times)
NXTSoundSensor ss = new NXTSoundSensor(SensorPort.S3);
SampleProvider sp = ss.getDBAMode();
float[] samples = new float[1];
ss.fetchSample(samples, 0);
if (samples[0] > 0.50f) { // For Nuno!
  return(true);
else {
  return (false);
} // BIG WTF
```

Feedback

```
final int FORWARD_TIME = 173;
final int SIDES = 4;

for (int side = 0; side < SIDES; side++){
   mLeft.forward();
   Thread.sleep(FORWARD_TIME);
}
NXTSoundSensor ss = new NXTSoundSensor(SensorPort.S3);
SampleProvider sp = ss.getDBAMode();
float[] samples = new float[1];
ss.fetchSample(samples, 0);
return samples[0] > 0.50f;
```

Its Final

Naming Numbers

- Its nice to use words instead of having to type numbers.
- We should use `Math.PI` instead of remembering 3.141592654...
- Its better to use a name that you define once (like `WHEEL_SIZE`).
- When you change the wheels, you only have to change one line of code.
- Define constants just like other fields, but in UPPER CASE.
- Use Comments to help the reader (you in a week).
- **Do not use static (global) objects!**
  Create them in a short `main`
  Pass objects to the constructor of other classes.

Behaviours

The Rodney Brooks subsumption architecture

We have to manage complex robot behaviour

- We managed robot movement using the navigation stack
- Now we need to combine many simple tasks
- To make a robot that appears intelligent
- To make a robot that plans to achieve its goals
- Better than Threads: Who's watching that Battery Voltage?

Subsumption

A guy with three active behaviours

- walking along (behaviour 1),
- eating (behaviour 2) and
- chatting (behaviour 3).
- ...who falls over - arms shoot forwards (behavior 4 subsumption).

It is vitally important that behaviours subsume each other.

Subsumption architecture

Plants do not grow towards the sun.

- The hormone Auxin promotes growth.
- The hormone Auxin is denatured by sunlight.
- Plants have no intelligence (Charles).
- Behaviours just get triggered.
- Plant cells grow when not in sunlight.
- **Responsive behaviours = Complexity.**

Subsumption

A Group of behaviours is intelligence

- No central controller.
- Behaviours have conditions on which they become active.
- Behaviours control different activities.
- When active they do their thing.
- They are suppressed by higher level behaviours.

Do you remember the last time *conditions* made you touch your face/hair.

The Behaviour Stack: Subsumption in leJOS

An `Arbitrator` manages a list of `Behavior` objects.

- Loops forever asking each `Behavior` object whether it wants to `takeControl`.
- Runs the highest priority `action` method in its own thread.

```
WHILE TRUE:
  // Build a list of waiting Behaviour actions
   ASK each Behavior if it wants to takeControl.

   IF no Behavior action is currently running THEN:
      RUN highest priority waiting Behavior
   ELSE IF there is a higher priority Behavior waiting
      // Notice that we NEVER STOP a currently active Behaviour
      CALL suppress on the currently running Behavior
```

The Behaviour Stack: Subsumption in leJOS

Subsumption - `suppress` and `action` run in different Threads

- CASE ONE: When a `Behavior` is running
  and a higher priority `Behavior` needs to take over
  …the `Arbitrator` keeps calling `suppress`
  …eventually this running `Behaviour` will get the message (and stop)
- CASE TWO: When no `Behavior` is currently running
  …the `Arbitrator` starts the highest priority waiting `Behavior`

The `suppress` method can be called any number of times (including none).

- If a `Behavior` has a short `action` then it can ignore `suppress` calls.
  The `action` will finish in good time anyway.
- A long running `action` method must notice `suppress`.
  The `action` method should finish early and exit gracefully.

Subsumption in leJOS: The `Behavior` interface

An `Arbitrator` manages a list of `Behavior` objects.

Every `Behavior` in leJOS must implement these three methods

- a `boolean takeControl()` that the `Arbitrator` calls to see if it wants control.
- an `void action()` that it wants to do when it has control.
- a `void suppress()` method the `Arbitrator` invokes to persuade the `Behavior` to finish quickly and exit gracefully.

Understanding a `Behavior`

A Single `Behavior` responds to a certain set of conditions

- It deals with some problem (e.g. Battery Voltage)
- It just gets some appropriate task done (e.g. Turning Left, Going Home)

leJOS lets us create many `Behavior` objects and think about them separately

- We let an `Arbitrator` object organise them for us.
- Each `Behavior` takes control of the robot when it is active.
- It **should** respond nicely when the `Arbitrator` tells it to stop...
- ...because another (more important) `Behavior` needs a turn.

Building a `Behavior`

Sharing is playing nicely

- We have a class for each type of `Behavior` that we need.
- A `Behavior` class may need access to the `Pilot`, or the `Navigator` or to sensors.
- A `Behavior` class may need to access some state control object.
- It is fine to store the (shared) objects as local variables in your `main` method.
- Pass each (shared) object (as needed) into the constructor for a `Behavior` object.
- Each `Behavior` object stores (a reference to) the shared object in a `private static` field.
- If only one `Behavior` needs to access an object then push the object down to a (`private static`) field in that class.
- Construct and store objects as close as possible to where they are needed.

An example - Behaviour controlled driving

```
public class Driver {
  public static void main(String[] args) {
   MovePilot pilot = getPilot(MotorPort.A, MotorPort.B, 60, 29);
   pilot.setLinearSpeed(200);
   Behavior trundle = new Trundle(pilot);
   Behavior escape = new Escape(pilot);
   Arbitrator ab = new Arbitrator(new Behavior[] {trundle, escape});
   ab.go();  // This never returns! It is a blocking call.
  }
  private static MovePilot getPilot(Port left, Port right, int diam, int offset) {
   BaseRegulatedMotor mL = new EV3LargeRegulatedMotor(left);
   Wheel wL = WheeledChassis.modelWheel(mL, diam).offset(-1 * offset);
   BaseRegulatedMotor mR = new EV3LargeRegulatedMotor(right);
   Wheel wR = WheeledChassis.modelWheel(mR, diam).offset(offset);
   Wheel[] wheels = new Wheel[] {wR, wL};
   Chassis chassis = new WheeledChassis(wheels, WheeledChassis.TYPE_DIFFERENTIAL);
   return new MovePilot(chassis);
  }
 }
```

An example Behavior - Keep on Trucking

```
public class Trundle implements Behavior {
  private MovePilot pilot;
  Trundle(MovePilot p) {
    this.pilot = p;  // Save the (shared) pilot in a field
  }
  // Start trundling and return control immediately.
  public void action() {
    pilot.forward();
  }
  // Since action returns immediately this is probably never called
  public void suppress() {}
  // Is it my turn?
  public boolean takeControl() {
    return true; // Yeah - we are SO eager
  }
}
```

An Example Behavior - Get Me Out of Here

```
public class Escape implements Behavior {
  private MovePilot turner; // The passed in shared pilot
  private EV3UltrasonicSensor us = new EV3UltrasonicSensor(SensorPort.S1);
  private SampleProvider sp = us.getDistanceMode(); // This sensor will just be used by us
  private Random rgen = new Random(); // just used by us (a new piece of Java?)
  private float[] distance = new float[1]; // just used by us - clever name huh!
  Trundle(MovePilot p) {
    turner = p;
  }
  public void action() {
    turner.travel(-50);
    turner.rotate((2 * rgen.nextInt(2) - 1) * 30); // right or left 30 degrees at random.
  }
  public void suppress() { } // Not sensible to suppress this Behavior. Ignore suppress.
  public boolean takeControl() { // Is it my turn?
    sp.fetchSample(distance, 0);
    return (distance[0] < 0.20f);  // See how this is short, but still easy to read?
  }
}
```

Project Extras and more advanced programming

State Based Behaviours

Your robot has to do one thing and then another

- The conditions for a `Behavior` may not just be sensors values
- Think of a game where you alternate moves.
  - A button press could disable all of my move behaviours.
  - Another button press makes them possible again.

Use a shared `State` object to disable behaviours

- (Only when `state.getState() == 'M'`). Behaviours: Move to Thing
- (Only when `state.getState() == 'P'`). Behaviours: Pick up Thing
- (Only when `state.getState() == 'I'`). Behaviours: Identify Thing
- (Only when `state.getState() == 'H'`). Behaviours: Take Thing Home

Sharing the State object: What sort of state am I in?

State is a kind of internal sensor - looking at the *state* of the robot

- Create a `State` class and then a `State` instance `state` in `main`.
- The `state` will record what the robot is trying to do.
- Share `state` with any `Behaviour` that depends on, or alters, the robot state.
- The `State` class can have a single `char` field with getters and setters.

```
public boolean takeControl() { // Check some condition
   return ... && state.getState == 'P'; //  AND it is our turn
}
public void action() { // Look for something
   ...
   state.setState(`I'); //  Got it! change to Identifying.
}
```

HARD: Using (shared) state for rising edge detection (separate claps)

```
public class State { // methods rather than a char field is better
  private boolean waitingForRising = true;
  public boolean isWaitingForRising() { return waitingForRising; }
  public void setWaitingForRising(boolean b) { waitingForRising = b; }
}
```

```
// wait for a loud sound after a quiet one
public class WFRise implements Behavior {
  private State sharedState;
  private float[] sound = new float[1];
  public boolean takeControl() {
    if (!sharedState.isWaitingForRising()) return false;
    sp.fetchSample(sound, 0);
    return (sound[0] > 0.5f);
  }
  public void suppress() {}
  public void action(){
    //  Do the "Heard a new clap" action
    sharedState.setWaitingForRising(false);
  } // constructor and sp def'n not shown
```

```
public class WFFall implements Behavior {
  private State sharedState;
  private float[] sound = new float[1];
  public boolean takeControl() {
    if (sharedState.isWaitingForRising()) {
      return false;
    }
    sp.fetchSample(sound, 0);
    return (sound[0] < 0.2f);
  }
  public void suppress() {}
  public void action() {
    sharedState.setWaitingForRising(true);
  } // constructor and sp def'n not shown
```

States and Personality

Changing from one state of certainty to another

- We can control the flow of behaviours using a shared state.
- Drawing a state diagram is a very useful process.
- Groups of behaviours may be self-contained, e.g. behaviours for finding food.
- A robot appears to have a certain personality when doing these behaviours.
- Of course, different personalities can share some behaviours.
- We can make a table of which behaviours are possible in each personality.

Worrying Pitfalls

`suppress` must make `action` stop

- After calling an `action` method no other `action` can start until this one finishes.
- Long running `action` methods that ignore calls to `suppress` break the system.
- `suppress` can set a field that is checked by `action` method, causing it to return.

Fast Methods

- If `action` never takes a long time then `suppress` can just return.
- `suppress` should always return quickly.
- `suppress` will be called multiple times if `action` takes a while to finish.

Extra Sensors, Extra Programs

Extra Sensors

The cyclingProfessor GitHub account

The Android app in the EV3Sensors repository, connects to a leJOS brick.

It does OpenCV image processing, continually finds QR codes and reads NFC tags.

The Readme.md in the repository tells you how to make this work.

- You can use it as a set of extra sensors for your brick.

The examples repository contains several interesting examples

The LejosExamples repository includes code for the leJOS end of EV3Sensors.

It also contains the (in progress) PCMapper program.

QR, NFC, FaceFinder and Proximity Sensor

EV3Sensors app internals - you do not NEED to know this to use QR codes

- `ChatFragment.sendMessage` sends a `String` to the brick.
- `ChatFragment.mHandler.handleMessage` processes messages from the EV3 in the (switch) case `MESSAGE_READ`.
- Messages from the EV3 are shown in a `ListView`.
- NFC tag and QR code data etc., are all be sent to the EV3 brick.
- There is example OpenCV blob detection in the app.
- You can easily detect faces using openCV with the app.
- Advanced: you can use all other phone sensors.

The PCMapper

Using the PC as an output or control device or even a *GPS Sensor*

- Using EV3Sensors you can use your mobile phone to read QR codes etc.,
- Your leJOS program can connect **at the same time** to a PC/laptop.
- You can use this (PC) to display, for example, the robot position on a map.
- You could use this as a way of updating an extended `OdometryPoseProvider`.
- Perhaps when you click the map, it sends an updated `Pose` to the robot.
- My example (PCMApper, EV3mapper) can be modified for your purposes.

Self Test

Questions

1. Are you ready?