SoftBank Robotics

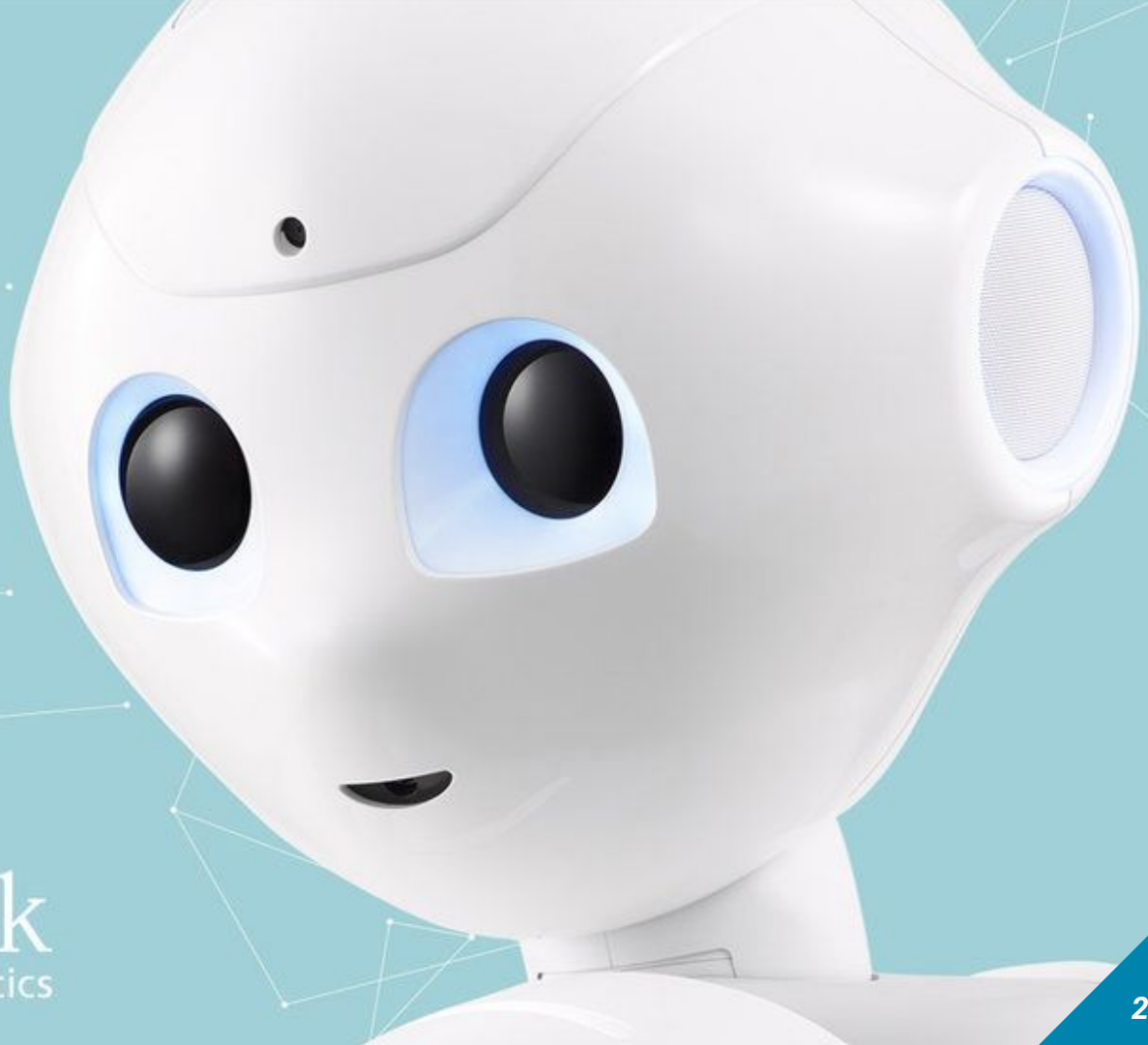
pepper

Pepper & the QiSDK

Let's make some apps!

December 2018

Day 1



SoftBank Robotics

SoftBank Robotics

pepper

Welcome!

The tech team

Feng Kai
Solution Consultant

Softbank Robotics China

Emile Kroeger
Robot App Expert

Softbank Robotics Europe

Software Team(Paris)

Agenda for the next 3 days

Day 1

Discover Pepper

Discover The QiSDK

Conversation

Day 2

Enter Interaction

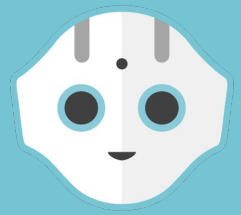
Asynchronous Programming

Navigation

Day 3

Tips & Tricks

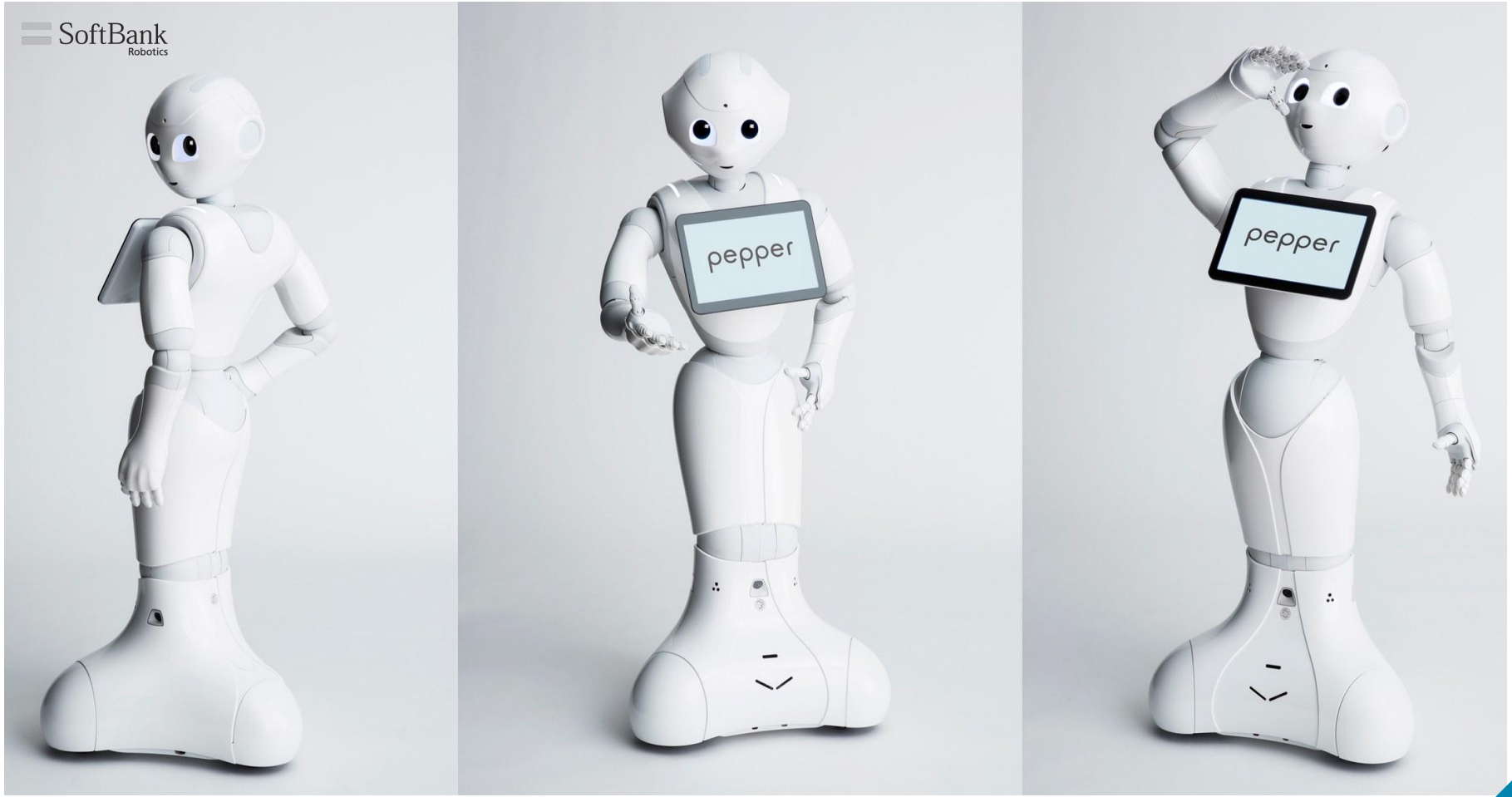
Making a full app



pepper

the Humanoid Robot

SoftBank Robotics



Standing:

working posture

- Standing,
  arms along the body
- Pepper is awake
  and ready to use

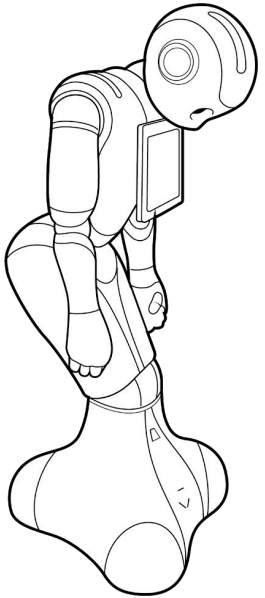
Rest:

safe posture

- Head down
- Knee and hip bent

Used when:

- Motors are off
- Rest mode
- Pepper is off



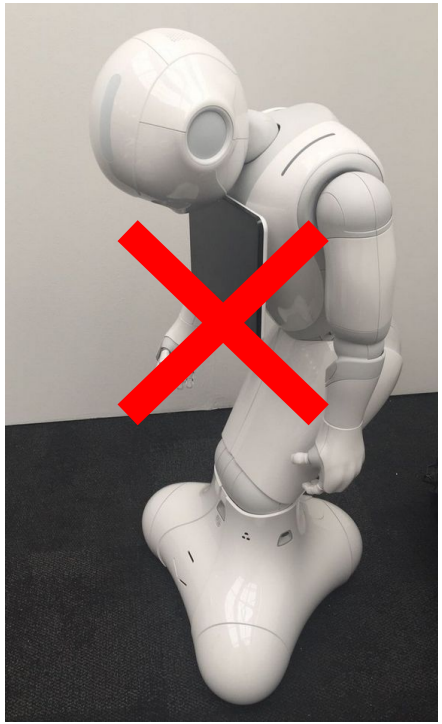
Communication Rules

DON'T

Please don't share any picture of Pepper:

- Sleeping
- Off
- In the box
- Stored
- In big group (army!)

Even during the training!



DO

Pepper is an alive, communicative, happy humanoid robot!

Active posing as if Pepper was speaking.

Scenes with/among people are strongly recommended.

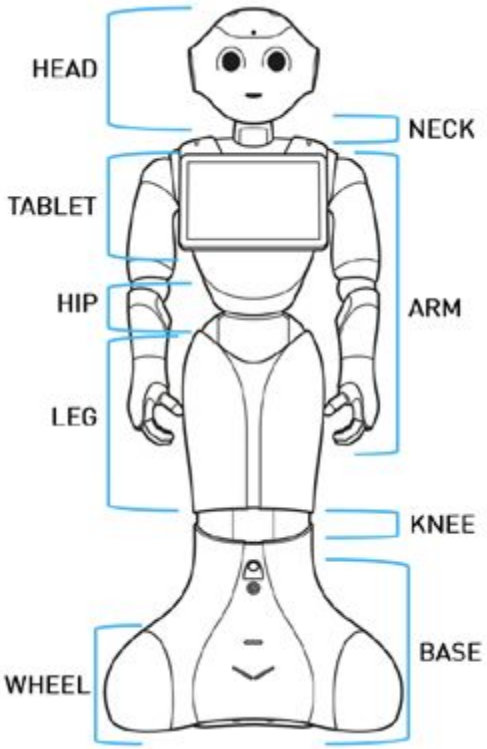
SoftBank Robotics

pepper

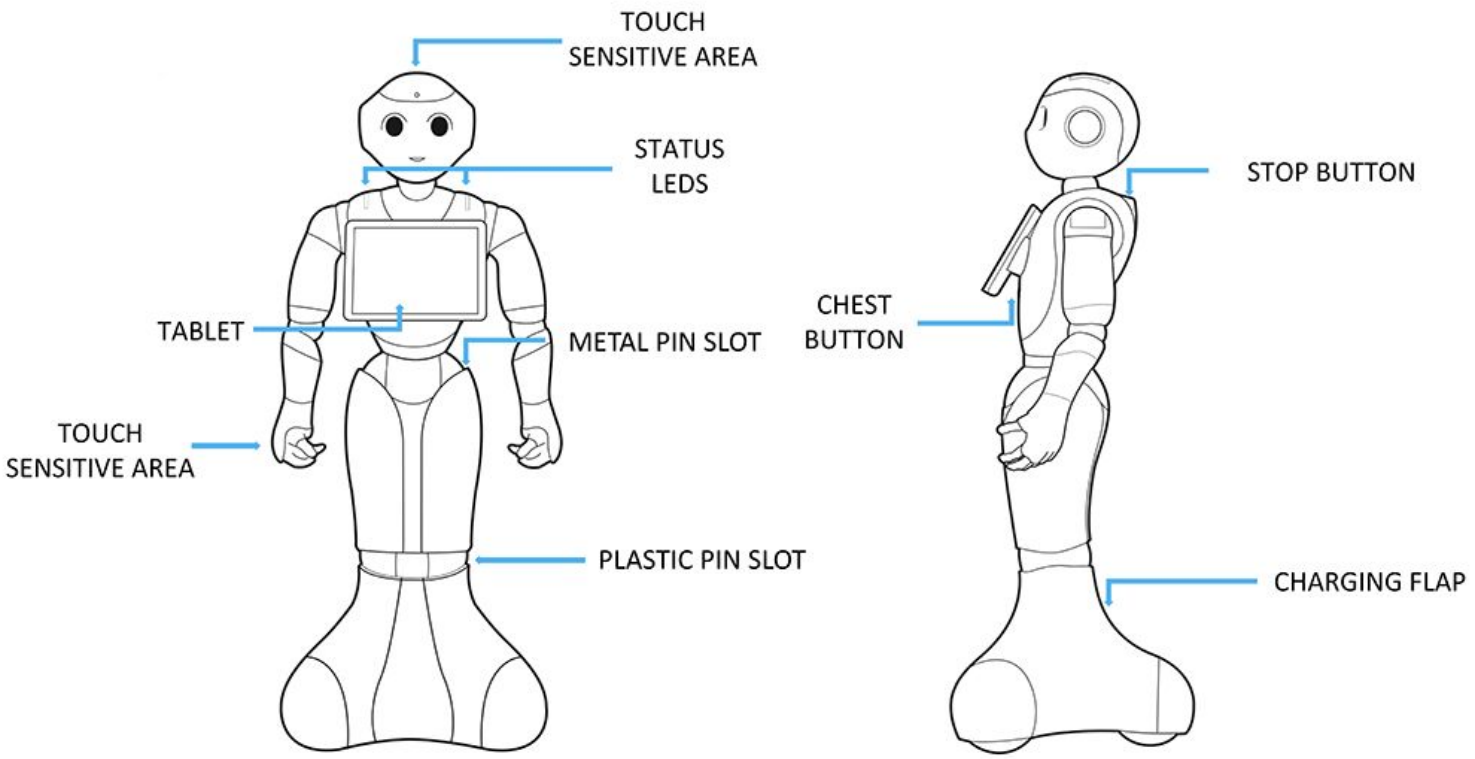
Hardware

Pepper's Anatomy

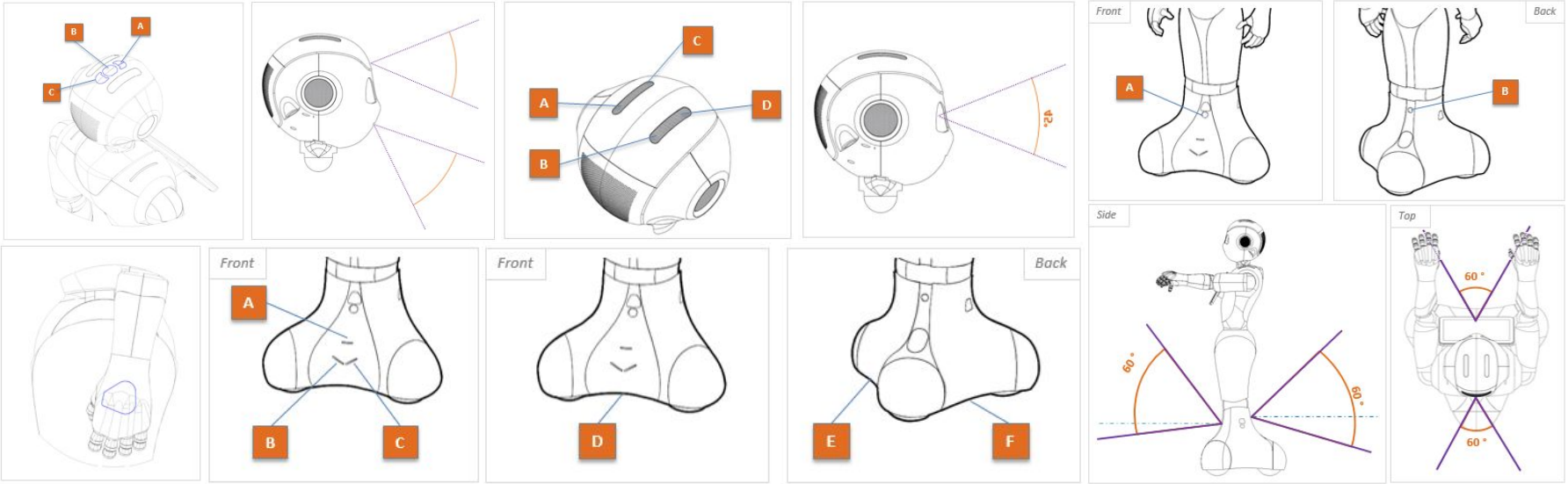
The body is divided into several parts:



Areas of interest | Pepper's Anatomy



Sensors | Pepper's Anatomy

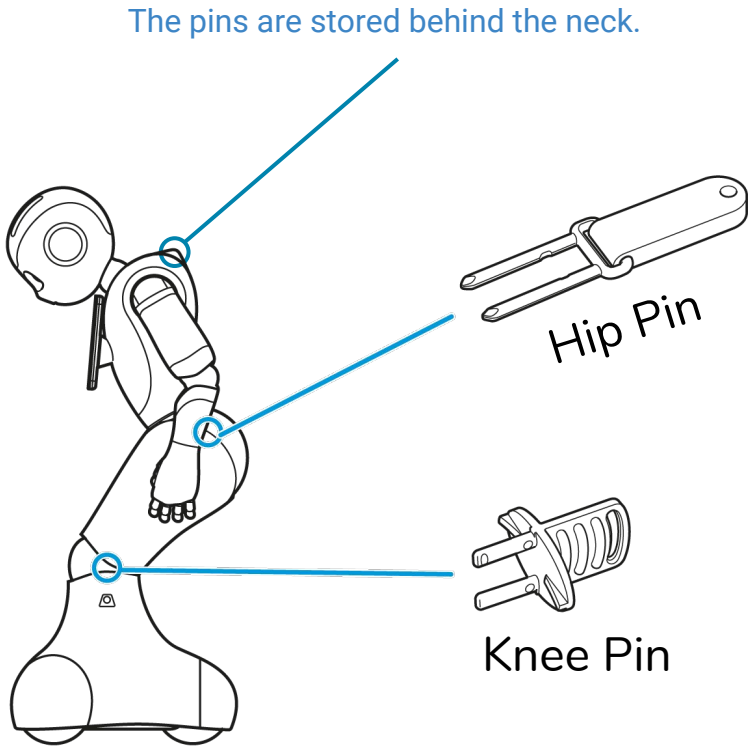


Brakes and Pins

The hip and knees have brakes
to prevent Pepper from falling over.

Use the 2 pins to release the brakes:

- When you put Pepper in his box
- For manually setting Pepper's posture
- To move or carry Pepper

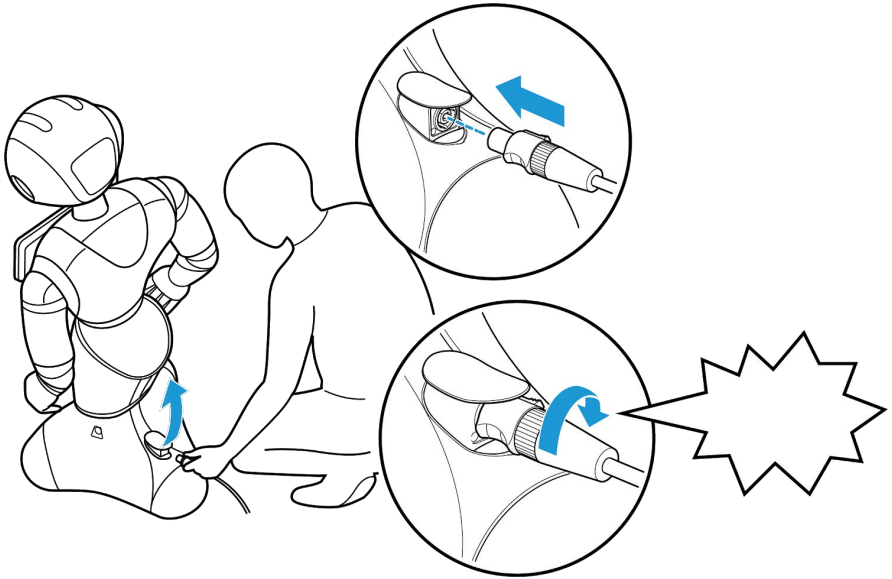


To charge Pepper:

1. Open the charging flap
2. Insert charger connector
3. Turn connector to the right until it clicks

Charging Flap = Mobility Security

When the charging flap is opened, the wheels' motors are deactivated.
=> Open it if you don't want Pepper to move around but still want to use him

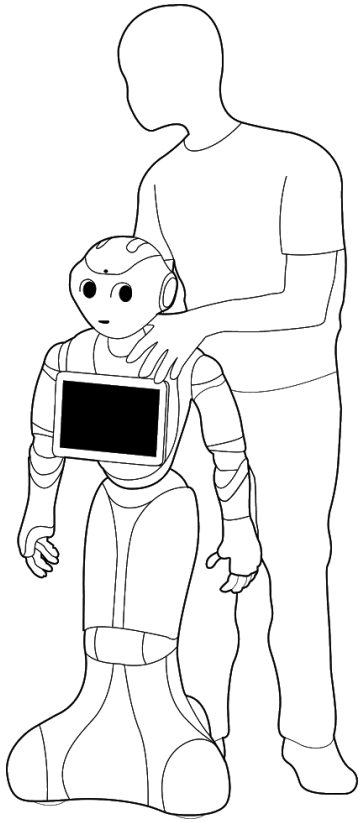


Charging duration:

80% in 3h30 - 100% in 8h

To move Pepper:

- Go to the Rest position
- Make sure the charging flap is opened
- Hold the robot
  - One hand on the shoulder
  - One hand on the hip
- Push it carefully



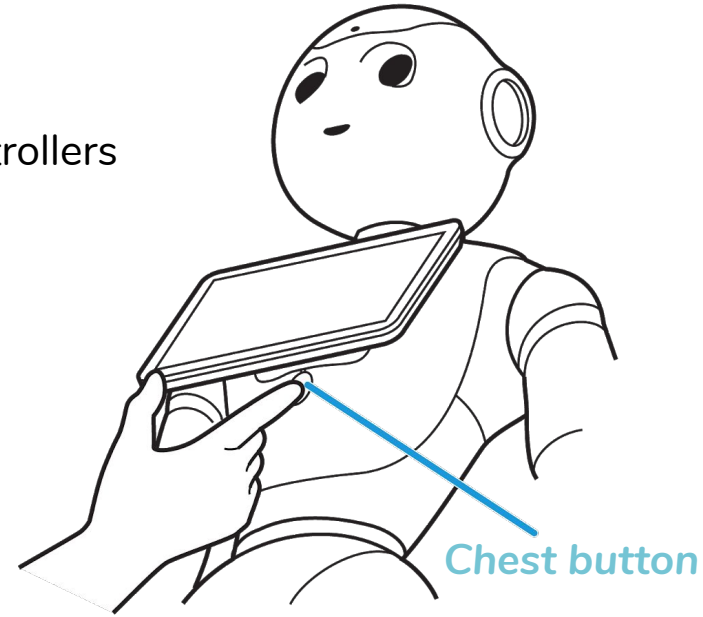
The chest button has multiple uses:

When Pepper is **OFF**:

- Press **once**: start Pepper
- Press & **hold**: check and flash the microcontrollers & start Pepper

When Pepper is **ON**:

- Press **once**: get status and notifications
- Press **twice**: Rest / Wake up
- Press and hold **3s**: turn Pepper OFF
- Press and hold **8s**: force switch OFF



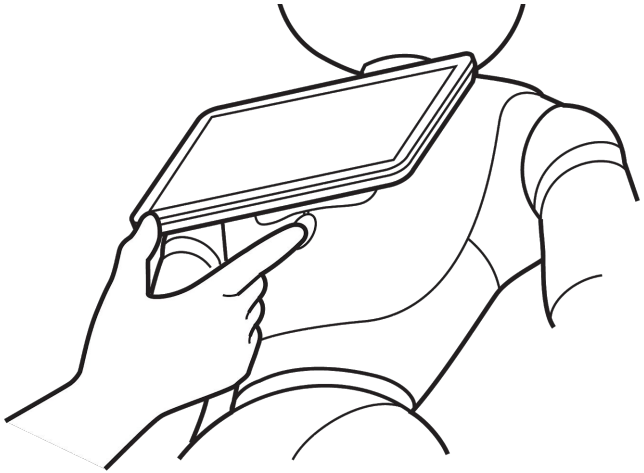
SoftBank Robotics

pepper

Interaction

Meeting Pepper

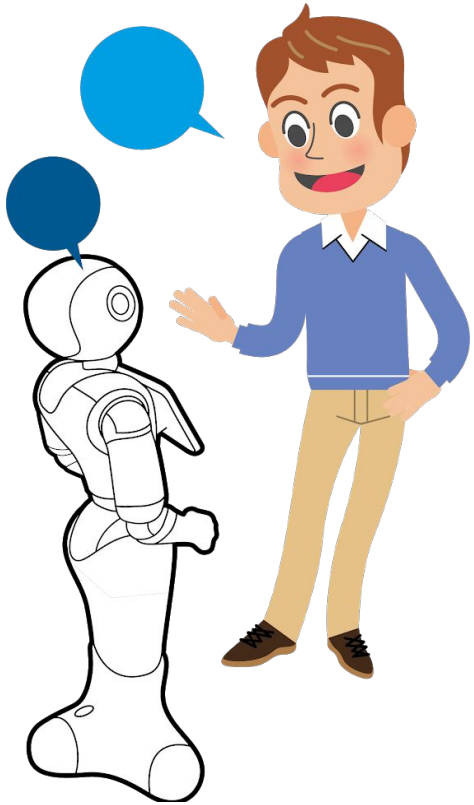
To boot, press once in the chest button under the tablet



- During the process:
  - Shoulder LEDs are blinking slowly
  - Ear LEDs act as a progress bar
- Booting takes approx. ***~60 seconds***
- When Pepper says:
  "OGNAK GNOUK"...this means he's on!

Basic awareness

- When your robot starts, it stands up and starts looking for people
- Pepper is now able to react to basic stimuli:
  - Movements
  - Tactile contacts
  - Human Presence
- The goal is to find a human and interact with him/her!



Interaction zones | *Talking to pepper*

< ~1.5m

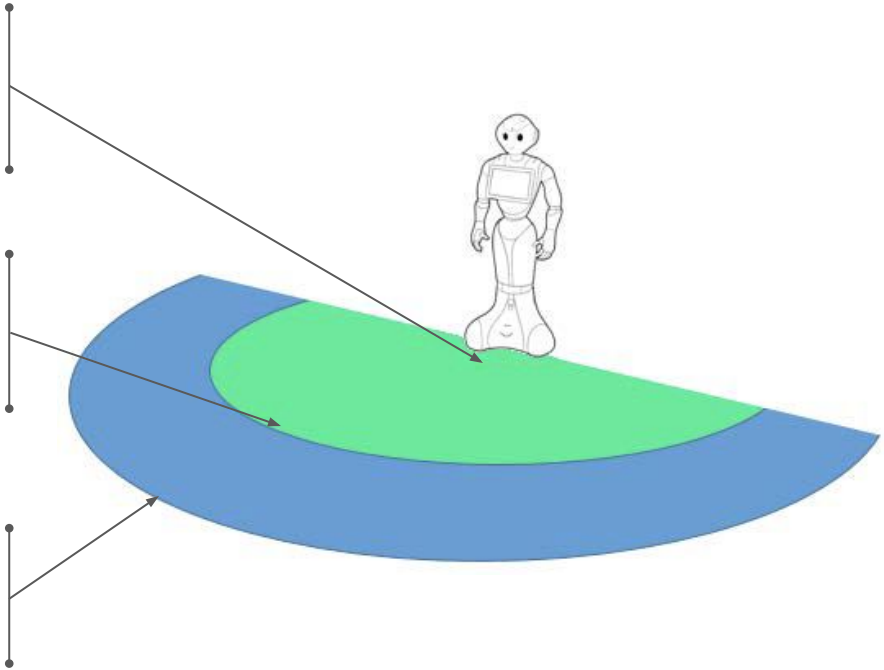
You are close enough to Pepper to have a conversation

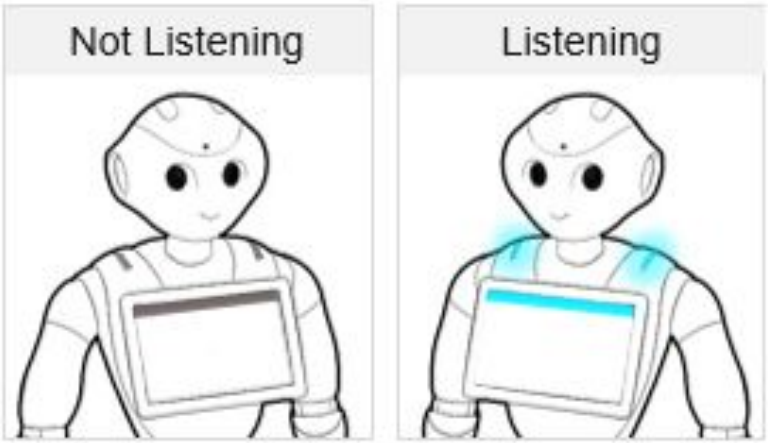
< ~3m

You are too far for a conversation, but you could hear Pepper speak

> ~3m

You are very far, Pepper sees you but you cannot hear each other.





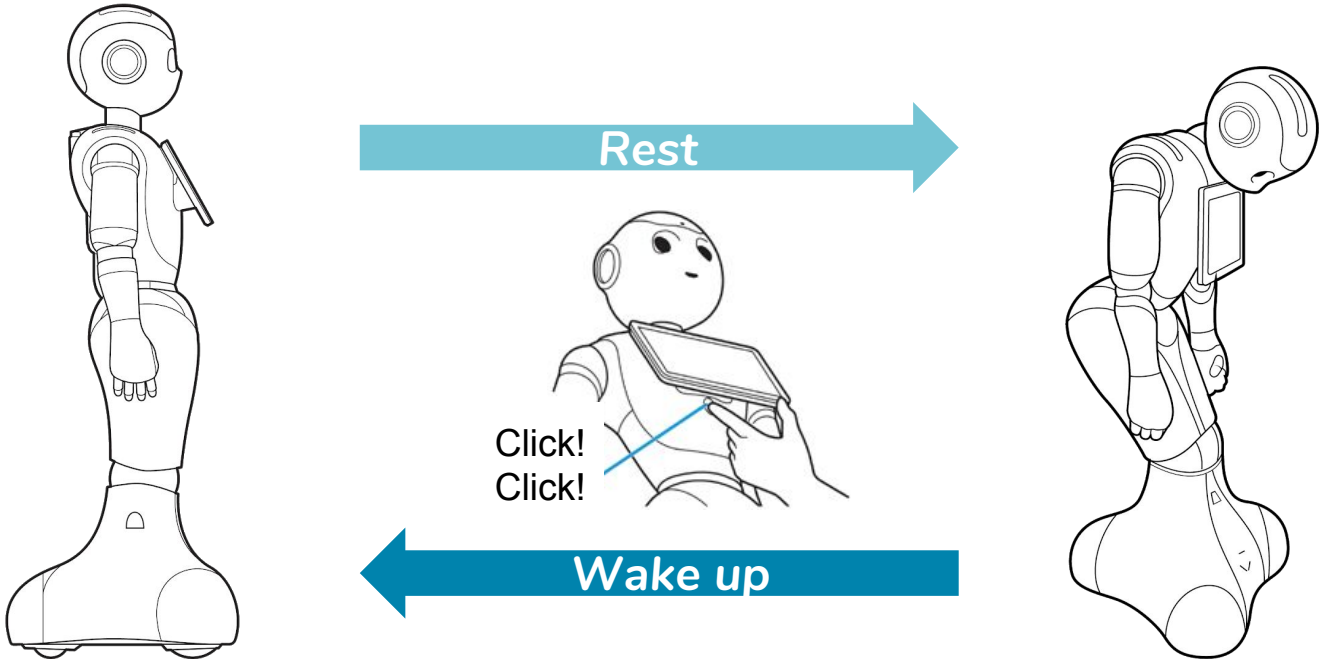
The **SpeechBar** and the **Leds** It helps to understand if the robot is listening, hearing something and what has been understood.

Hearing Human Voice

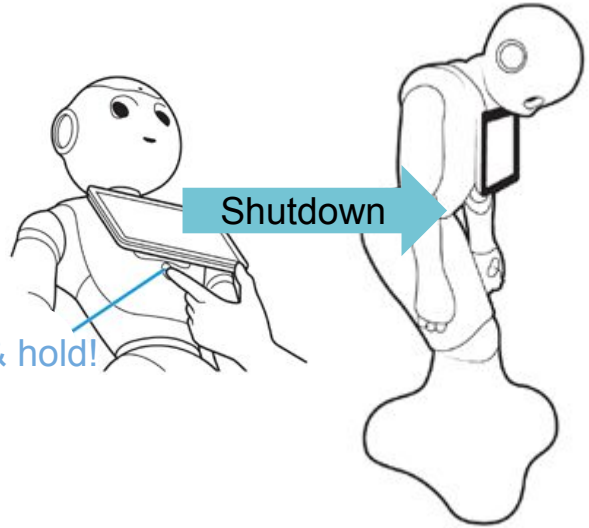
Not Understood

Understood

To change between Rest & Awake, double-click the chest button!



- Press the chest button for 3s,
  until the robot says “GNUK GNUK”
- The robot will go to its “Rest” position
- The process with take ~30 seconds



SoftBank Robotics

pepper

Android & Pepper

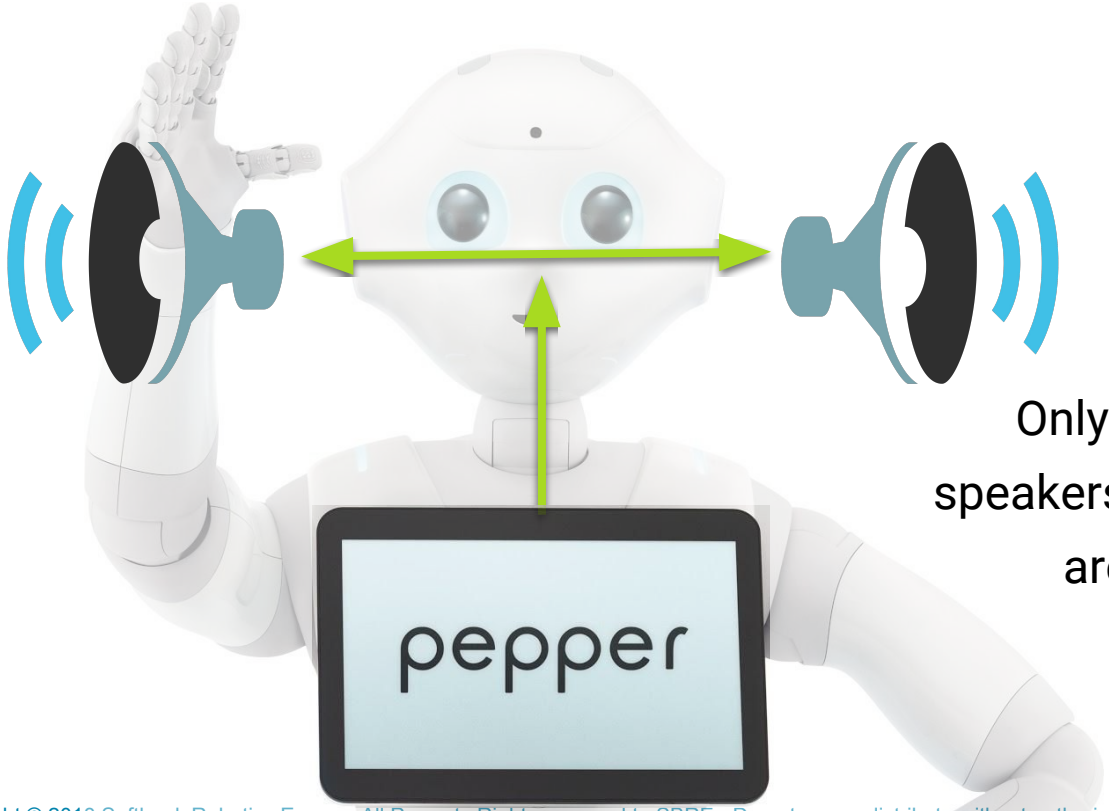
Pepper's Head

- Running NAOqi OS *(based on Linux)*
- CPU: 1.91 GHz quad-core Atom E3845
- Wi-Fi

Pepper's Tablet

- Running Marshmallow
- Certified GMS (CTS)
- CPU: 1.3 GHz quad-core ARM
- Wi-Fi, Bluetooth
- DDR3 SDRAM: 1GB *(512MB * 2)*
- 15GB
- Size: 10" Type: IPS Resolution: 1280x800
- Multitouch

Audio Output



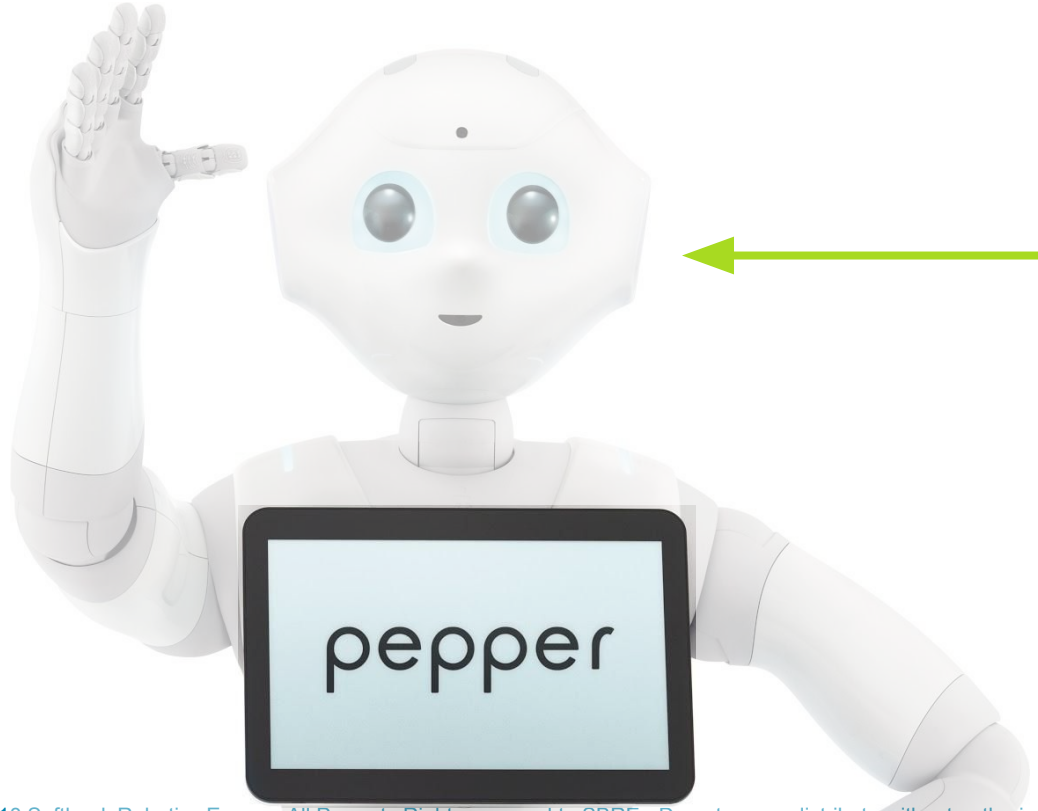
Only Pepper’s speakers on the head are used.

Audio Input

The QiSDK APIs

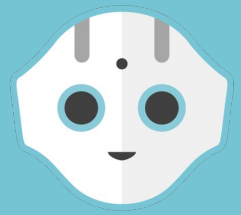
(Listen, Chatbots...)

use Pepper's microphones



SoftBank Robotics

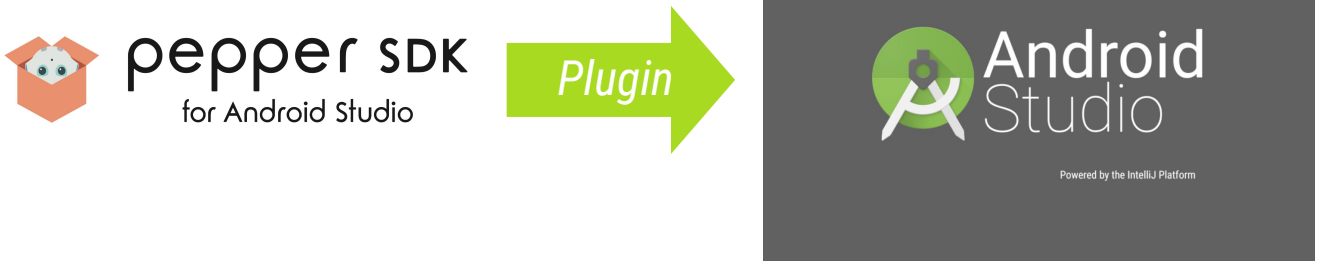
pepper



Android Studio

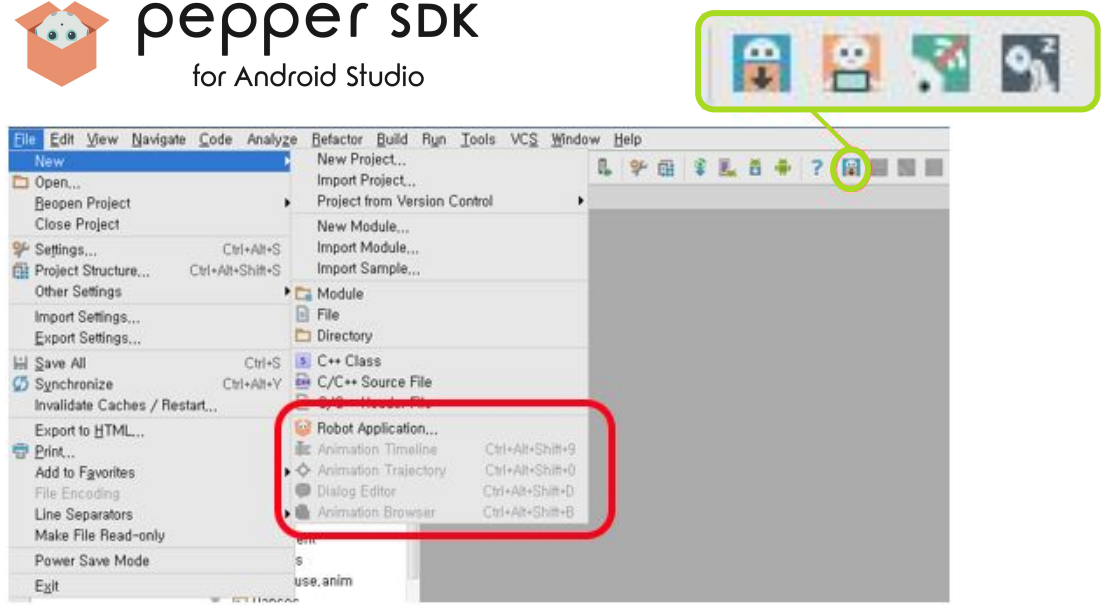
PLUG IN

Make apps for Pepper using Android Studio.



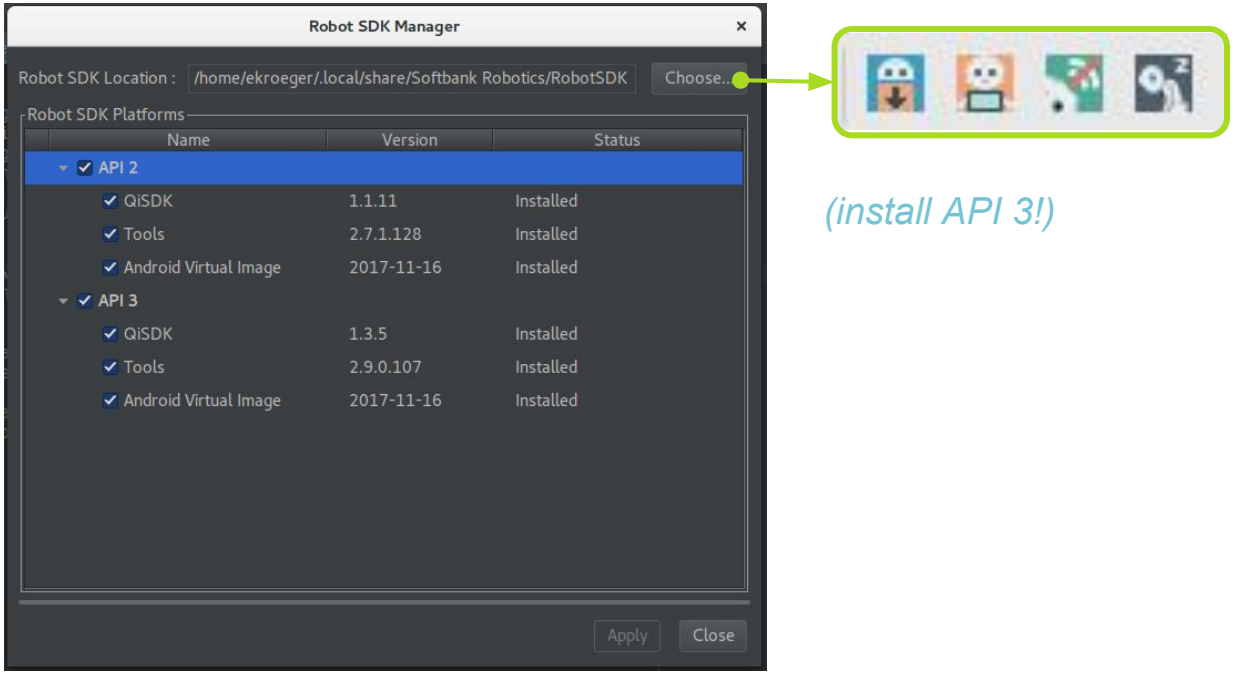
- **Plugin**

pepper SDK
for Android Studio



Android Studio Plugin

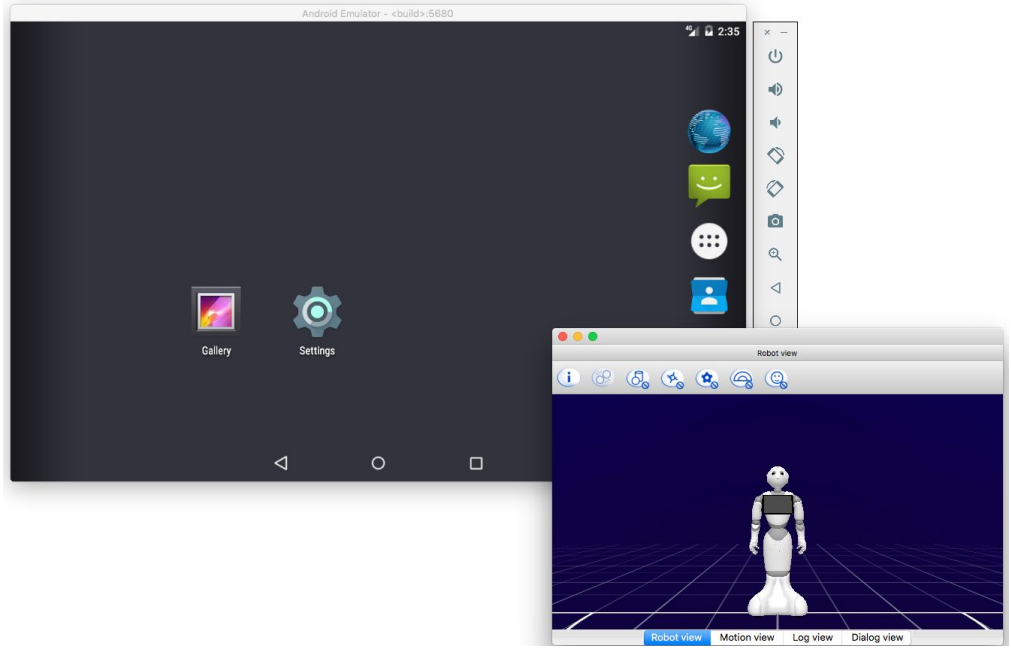
- **Plugin**



(install API 3!)

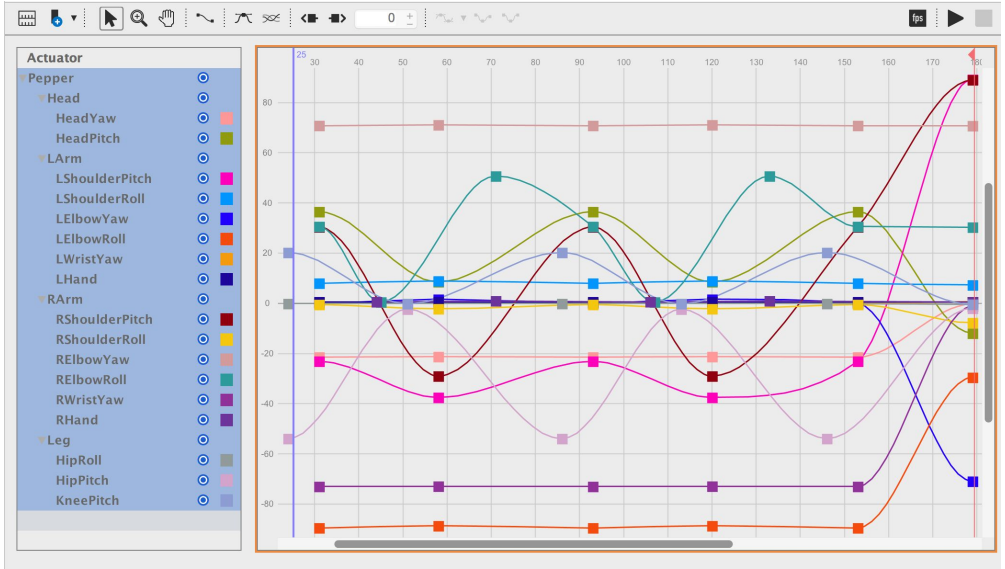
Android Studio Plugin

- Plugin
- ***Virtual Robot + Android virtual device***



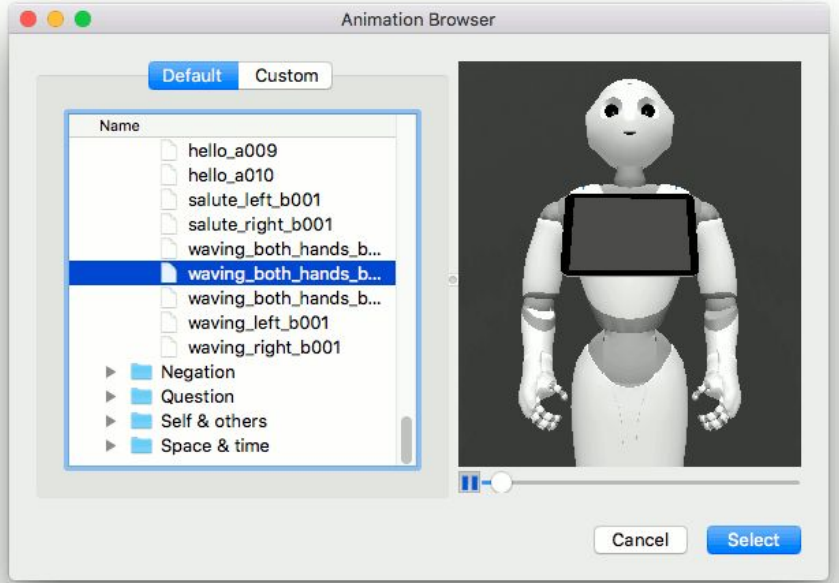
Android Studio Plugin

- Plugin
- Virtual Robot +
  Android virtual device
- ***Animation Editor***



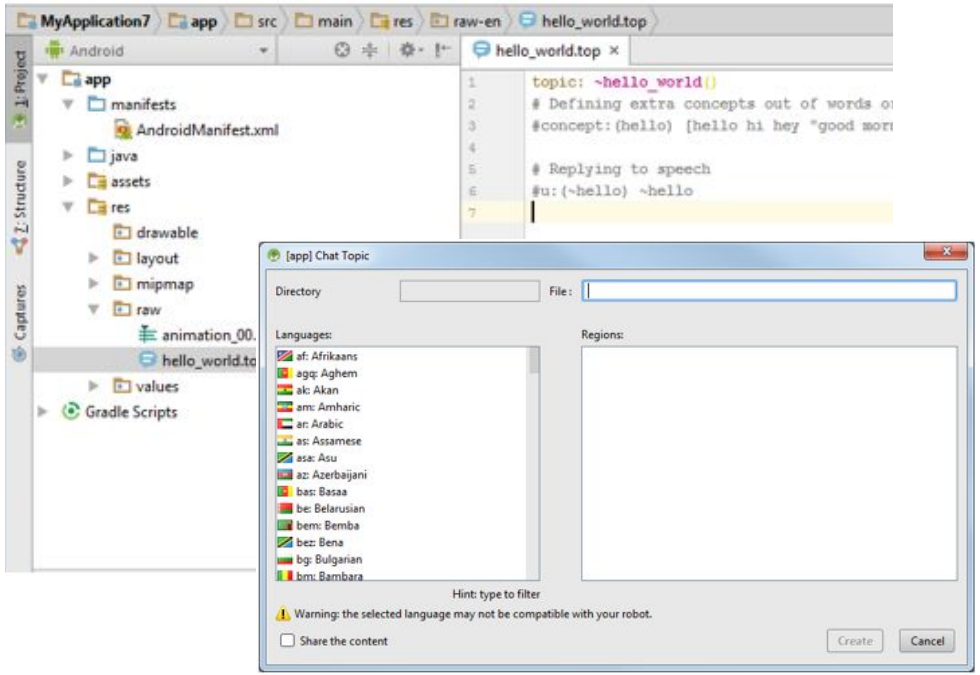
Android Studio Plugin

- Plugin
- Virtual Robot +
  Android virtual device
- Animation Editor
- ***Animation Library***



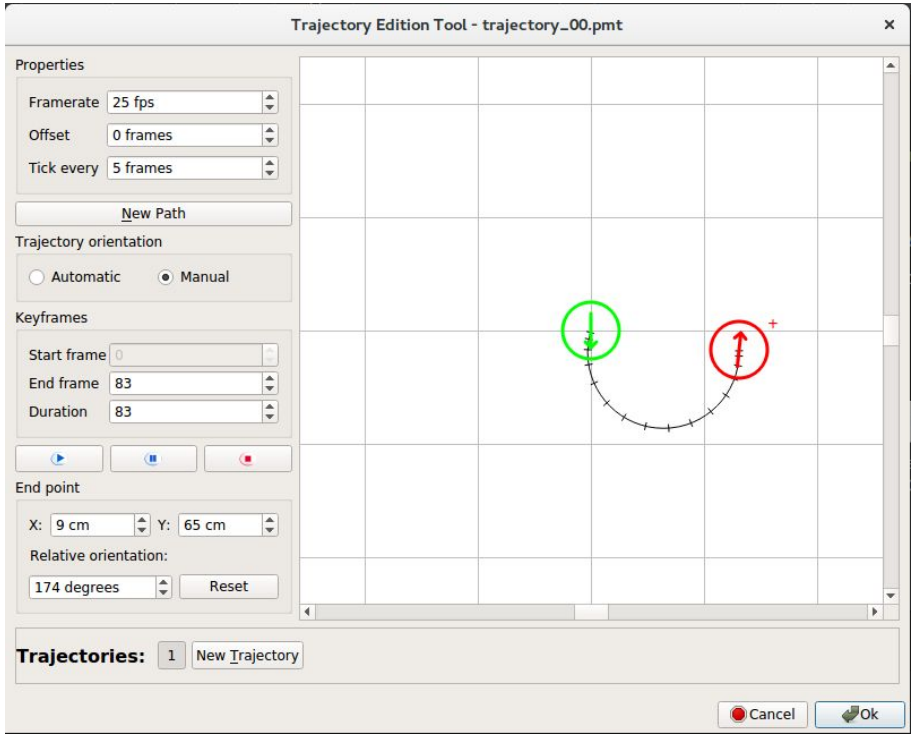
Android Studio Plugin

- Plugin
- Virtual Robot +
  Android virtual device
- Animation Editor
- Animation Library
- ***QiChat Editor***



Android Studio Plugin

- Plugin
- Virtual Robot +
  Android virtual device
- Animation Editor
- Animation Library
- QiChat Editor
- ***Trajectory Editor***



Supported programming languages

Java 6, 7 or 8

https://developer.android.com/studio/write/java8-support

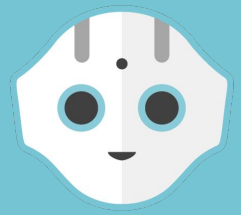
Kotlin

https://developer.android.com/kotlin/

https://kotlinlang.org/docs/reference/

SoftBank Robotics

pepper



Hands-on:
Discovery of QiSDK

Let's make a very basic *"Hello World"* app

To learn about:

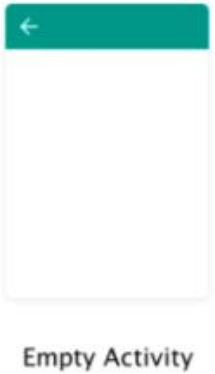
- The Robot Lifecycle
- QiSDK Actions

Step 1: Create the project

Create a new project with an Empty Activity

> use:

- minSdkVersion : 23 - Marshmallow
- targetSdkVersion : 27 - Oreo ***(latest stable)***



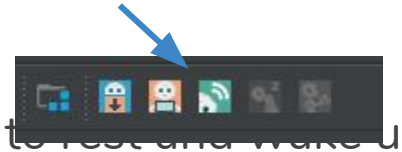
Empty Activity

Step 2: test our connection to Pepper

Before we anything, let's first ***Connect to Pepper!***

You will need to:

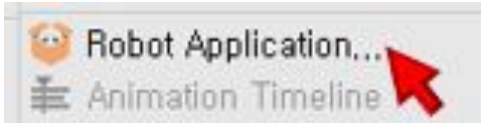
- Enable developer mode on Pepper's tablet (settings > kernel)
- Enable ADB in Developer Options
- Get Pepper's IP address (the robot's, not the tablet)
- Use the Plugin's "**Connect**" button



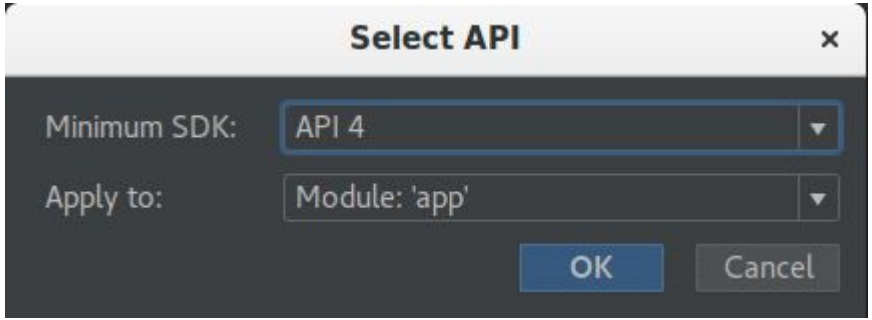
Then, try getting pepper to rest and wake up!

Make it a Pepper application:

File -> New -> Robot Application



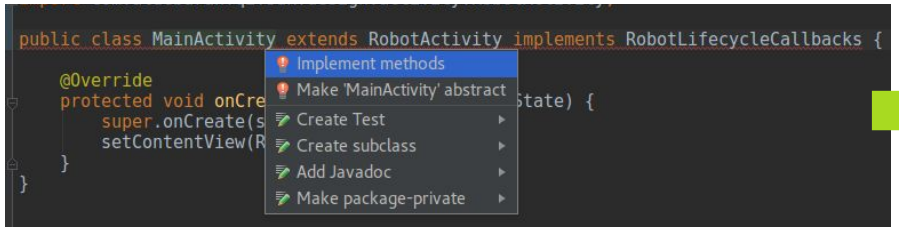
- Use ***SDK 4***:

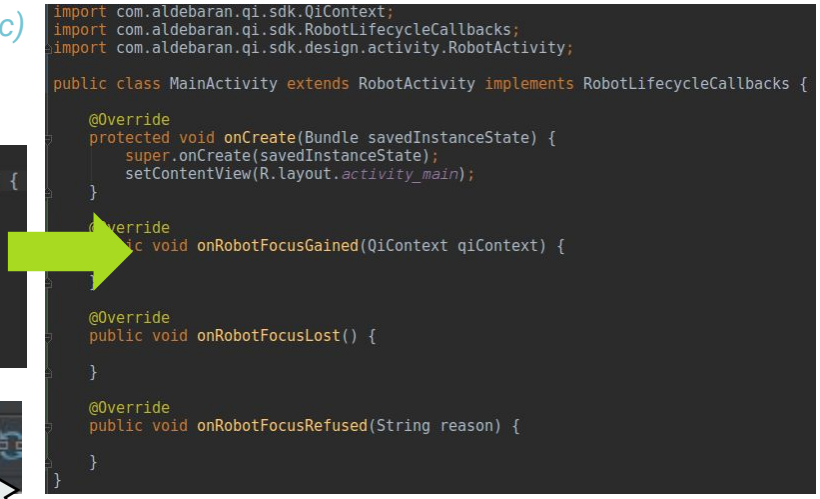


Step 4: RobotActivity

Your main activity must inherit from **RobotActivity**, and implement **RobotLifecycleCallbacks**:

Note: *you don't need to copy this code, just add the classes and then use Alt-Enter (⌥+Enter on mac) to import and autocomplete (as often in android)*





```
import com.aldebaran.qi.sdk.QiContext;
import com.aldebaran.qi.sdk.RobotLifecycleCallbacks;
import com.aldebaran.qi.sdk.design.activity.RobotActivity;

public class MainActivity extends RobotActivity implements RobotLifecycleCallbacks {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public void onRobotFocusGained(QiContext qiContext) {

    }

    @Override
    public void onRobotFocusLost() {

    }

    @Override
    public void onRobotFocusRefused(String reason) {

    }
}
```

Note: *if autocomplete doesn't suggest anything, do a gradle sync:*

Step 5: Plug in “say’

Now let’s get pepper to talk !

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withText("Hello everybody!")
            .build();
    say.run();
}
```

Note: on your first connection to the robot, you will have to allow ADB debugging on the tablet *(for each computer)*

Register your activity so it connects to the robot

Create and run a “say” action

WE WON’T CONTINUE UNTIL YOUR PEPPER SAYS HELLO :)

So what is going on here?

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withText("Hello everybody!")
            .build();
    say.run();
}
```

Activity Lifecycle

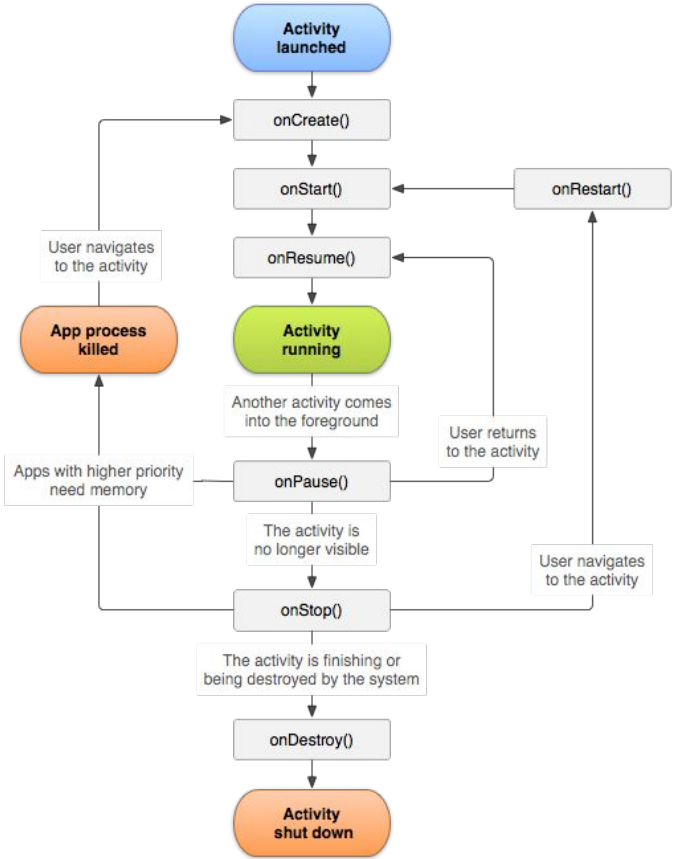
Actions

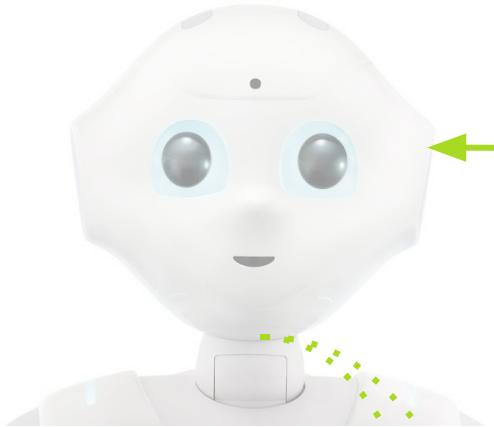
Let’s step back

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withText("Hello everybody!")
            .build();
    say.run();
}
```

Activity Lifecycle

Actions





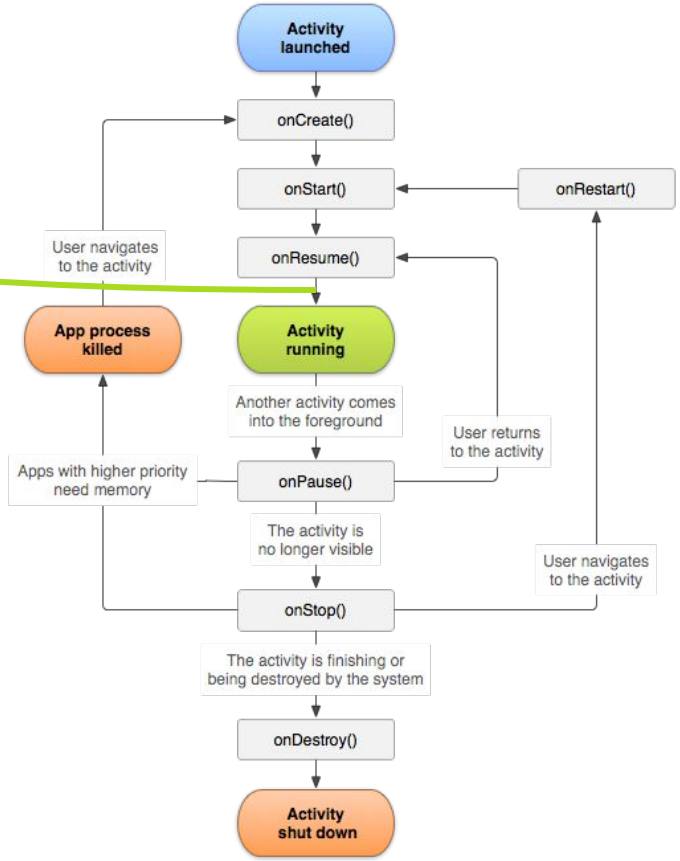
1

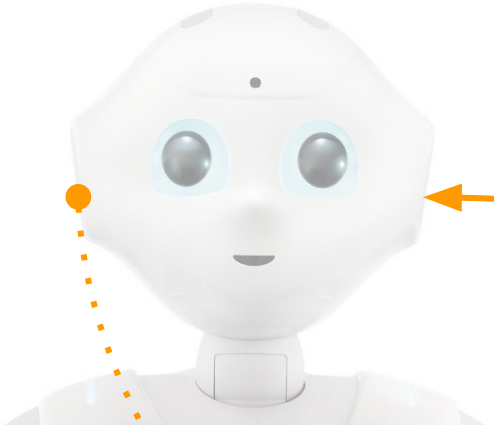
2

onRobotFocusGained

onRobotFocusRefused

onRobotFocusLost





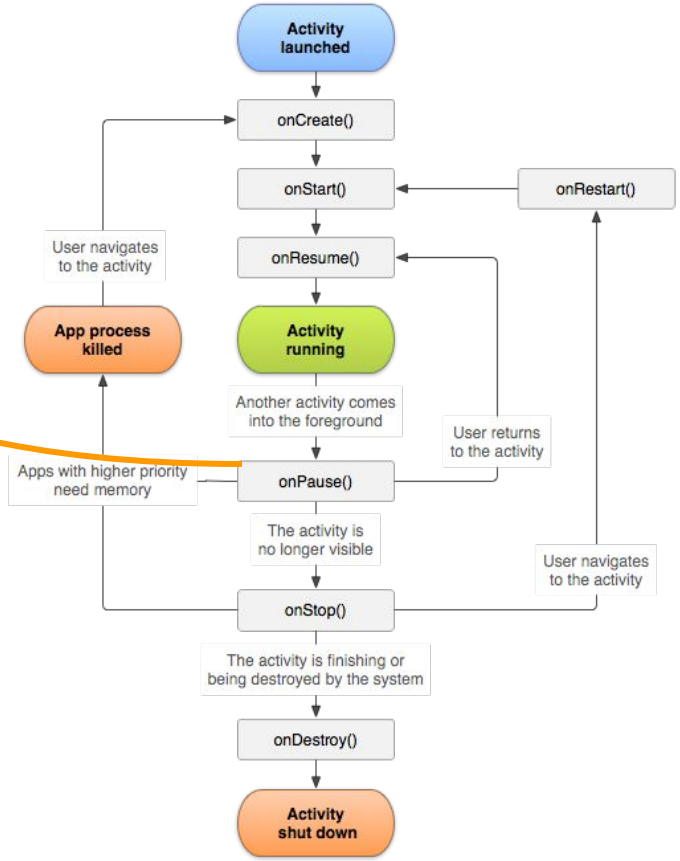
3

4

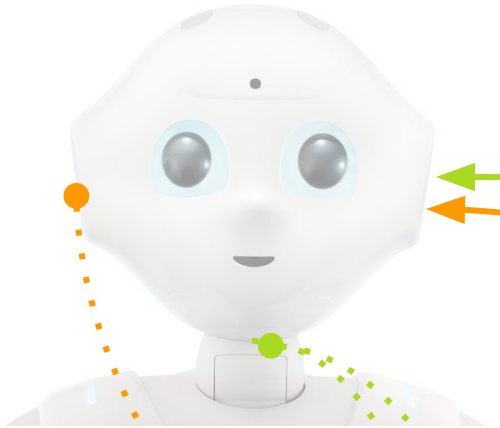
onRobotFocusGained

onRobotFocusRefused

onRobotFocusLost



```
public class MyActivity implements RobotLifecycleCallbacks
```



1

3

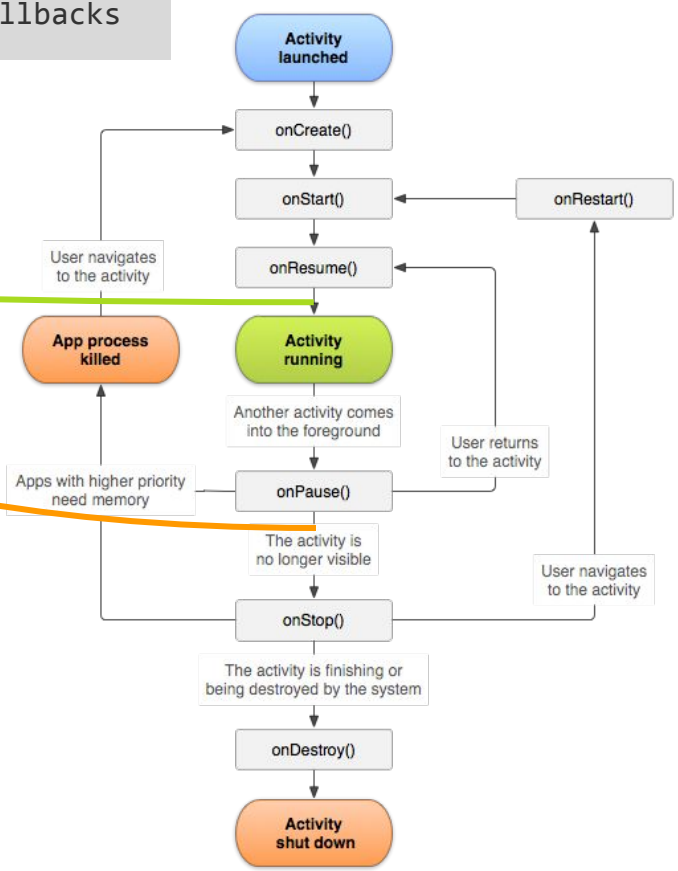
2

4

onRobotFocusGained

onRobotFocusRefused

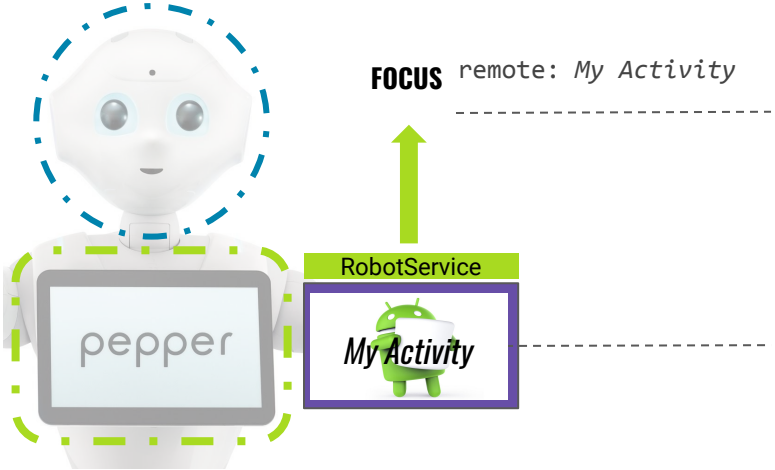
onRobotFocusLost



`onRobotFocusGained`

It provides a `QiContext` that allows you to:

- create actions,
- create resources for actions,
- retrieve robot services.

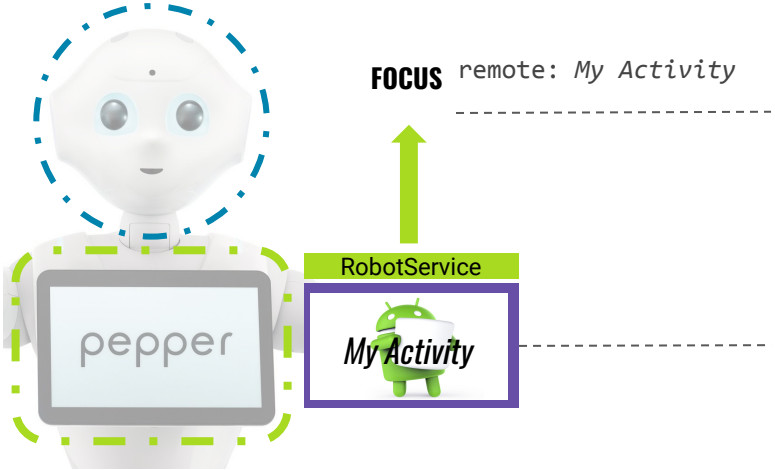


Managing the Focus

`onRobotFocusGained`

It provides a `QiContext` that allows you to:

- create actions,
- create resources for actions,
- retrieve robot services.



`onRobotFocusRefused`

It happens when the robot is in an unstable state and cannot give the focus.

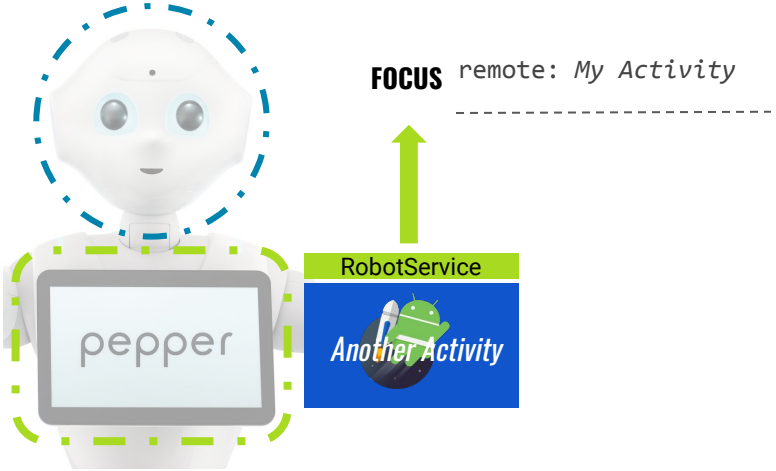
The `reason` is given as parameter.

Managing the Focus

`onRobotFocusGained`

It provides a `QiContext` that allows you to:

- create actions,
- create resources for actions,
- retrieve robot services.



`onRobotFocusLost`

Called when the `Activity` owning the robot focus, becomes not visible on the UI (even partially).

When this callback is called:

- the `Activity` cannot run actions on Pepper until it regains the robot focus
- If an action was running, the action execution will stop and an exception will be raised
- listeners will not be removed so they should be removed manually.

Addendum: on Destroy

Our Hello World is still messing one thing: `QiSDK.unregister`

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);
}

@Override
public void onDestroy() {
    QiSDK.unregister(this, this);
    super.onDestroy();
}
```

How about actions?

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withPhrase("Hello everybody!")
            .build();
    say.run();
}
```

Activity Lifecycle

Actions

What is an action?

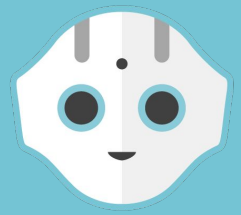
An **action** is what **makes Pepper do** things.

You use an action **Builder** to create an action **object**, passing needed configuration (a text in this example).

The action **object** can then be **run** on Pepper.

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
   Say say = SayBuilder.with(qiContext)
           .withText("Hello everybody!")
           .build();
   say.run();
}
```

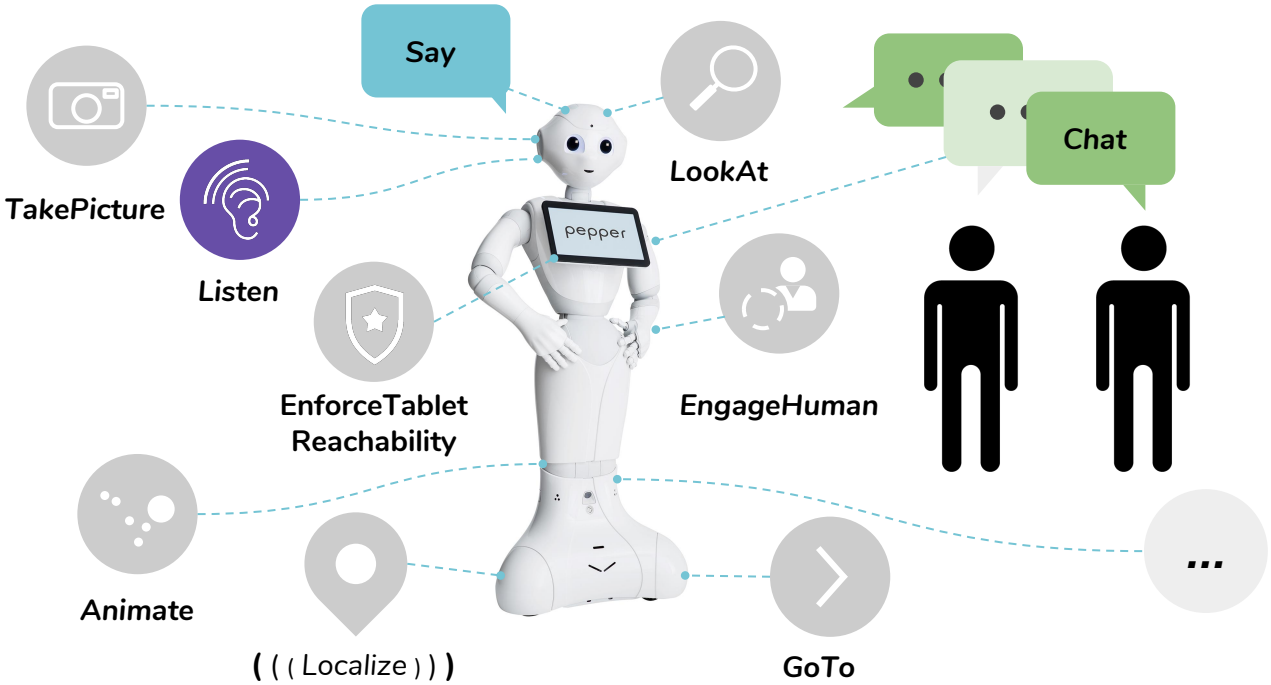
The QiSDK provides many other actions for Pepper:



Theory:
Conversation

QiSDK APIs: Conversation

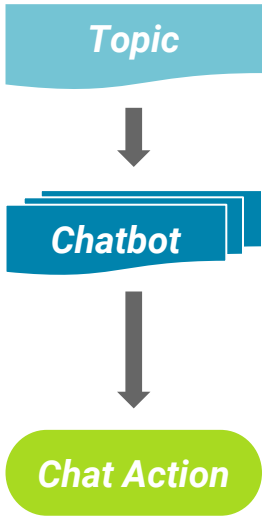
Let's look at [the documentation on Conversation]



QiSDK APIs: Conversation

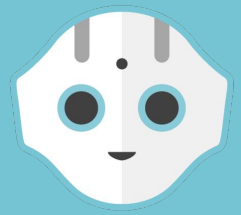
There are 3 concepts to understand:

- **Topic**
  - This is a .top file created in the resources folder of the project which is built using the QiChat syntax
- **QiChatbot**
  - A type of chatbot built into the SDK which uses the topics above. There are others you can use and you can use multiple 3rd party chat bots together if required
- **Chat Action**
  - The action which gives Pepper the ability to speak.
    When this action is running, the associated chatbots are active (Pepper listens or speaks).



SoftBank Robotics

pepper



Hands-on:
QiChatbot

Now let's make something more complex

- Create a new project as we did earlier…

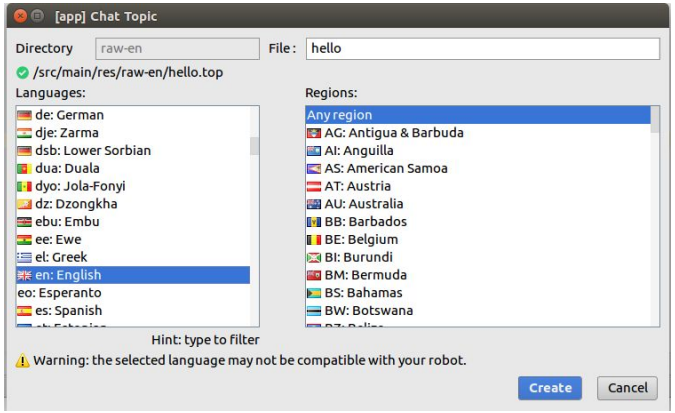
Empty Activity

- File > New project…
- Target latest APIs, make sure Marshmallow *(23)* is the min.
- Robotify the project *(File > New > Robot Application)*
- Open the MainActivity and implement the **callbacks** and extend **RobotActivity** as before

Let's start:

1. Create a .top file in the project
   a. File > New > Chat Topic
2. Call this 'hello' and give it the English language
3. The IDE will generate a basic conversation for us to play with.

/!\ There is a bug in the current version, you need to rename the `raw-en` folder to `raw`



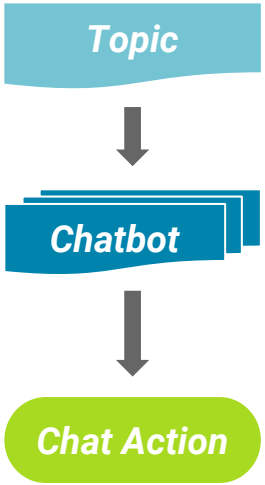

```
topic: ~hello()
# Defining extra concepts out of words or group of words
concept:(hello) [hello hi hey "good morning" greetings]

# Replying to speech
u:(~hello) ~hello
```

- The name of the topic
- A concept is a list of words or phrases which apply to an idea. This is important to make speech feel more natural.
- The robot response
- The human input

Build this Chat using the Topic and Chatbot

```
@Override
public void onRobotFocusGained(QiContext qiContext)
{
   //Create the topic
   Topic topic = TopicBuilder.with(qiContext)
           .withResource(R.raw.hello)
           .build();

   //Create the chatbot
   QiChatbot qiChatbot = QiChatbotBuilder
           .with(qiContext)
           .withTopic(topic).build();

   //Create the Chat
   Chat chat = ChatBuilder
           .with(qiContext)
           .withChatbot(qiChatbot).build();

   chat.run();
}
```

Let's implement this:



Once the chat is running, you can speak to Pepper.

Now that we have a topic, let's make it a bit more interesting

```
u:(I like _[tea coffee]) Good to know $drink=$1

u:(My name is _*) Pleased to meet you $1
$name=$1

u:(Do you know me?)
 ^first[
   "Yes, you are $name and you like $drink"
   "Yes, you are $name"
   "I know you like $drink"
   "No, I don't know you"
 ]
```

Using **underscore _** creates a special **variable**, **$1**, that you can assign to another variable

```
u:(I like _[tea coffee]) Good to know $drink=$1

u:(My name is _*) Pleased to meet you $1
$name=$1

u:(Do you know me?)
 ^first[
   "Yes, you are $name and you like $drink"
   "Yes, you are $name"
   "I know you like $drink"
   "No, I don't know you"
 ]
```

Use it with the **wildcard *** to get any inputs.

Use **^first** to get the first line with no empty variables.

Now let's add an icebreaker, using a **Bookmark**

In your topic:

```
proposal: %ICEBREAKER
   Hi I'm Pepper!
```

In your java (in `onRobotFocusGained`, before your `chat.run`):

```
Bookmark icebreaker = topic.getBookmarks().get("ICEBREAKER");
chat.addOnStartedListener(() -> chatbot.goToBookmark(
       icebreaker,
       AutonomousReactionImportance.HIGH,
       AutonomousReactionValidity.IMMEDIATE));
chat.run();
```

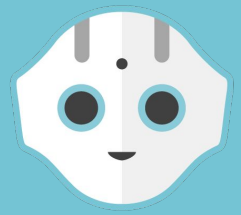
You can also get a callback when a bookmark is reached:

In your topic:

```
u:(goodbye {pepper})
   goodbye %FINISH_APP
```

In your java:

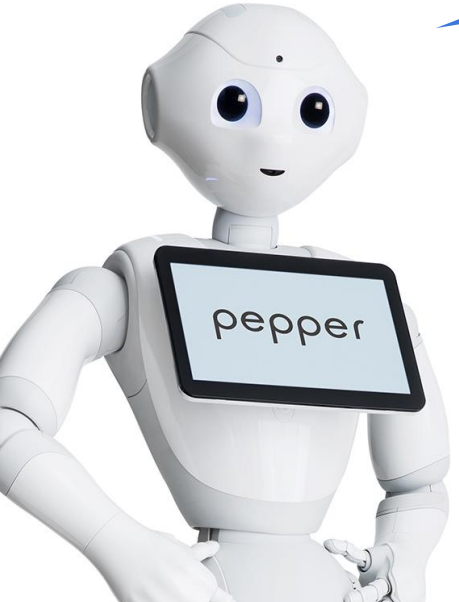
```
chatbot.addOnBookmarkReachedListener((bookmark) -> {
   if (bookmark.getName().equals("FINISH_APP")) {
       finish();
   }
});
chat.run();
```



Challenge:
Guess the Number!

Challenge:

Make a little game, where you make Pepper guess a number:



Think of a number and tell me when you're ready.

I'm ready !

Is it ten?

No it's smaller

How about five?

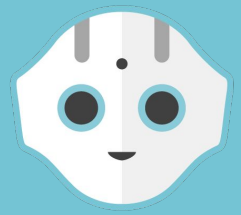
Still smaller

Three then?

No, larger

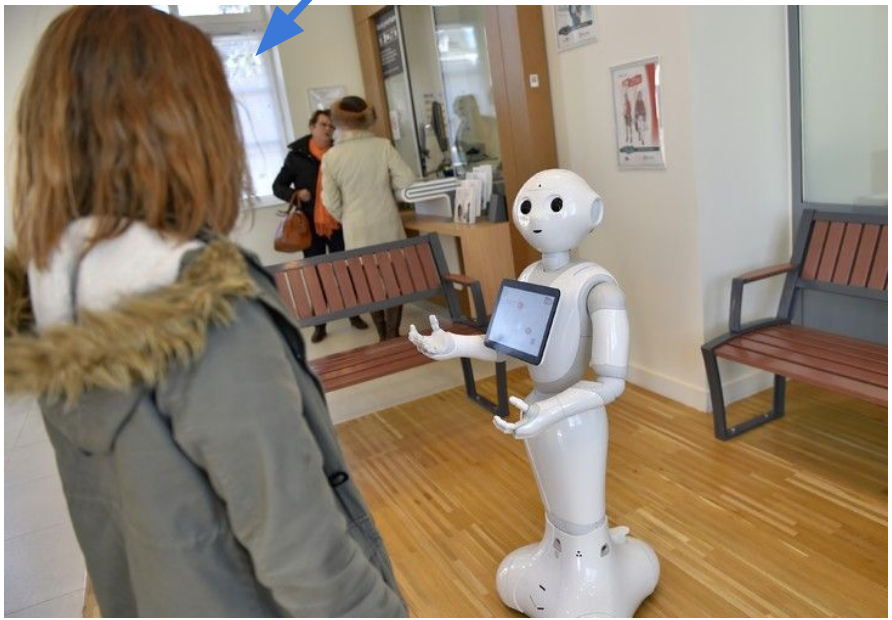
Four?

That's right!

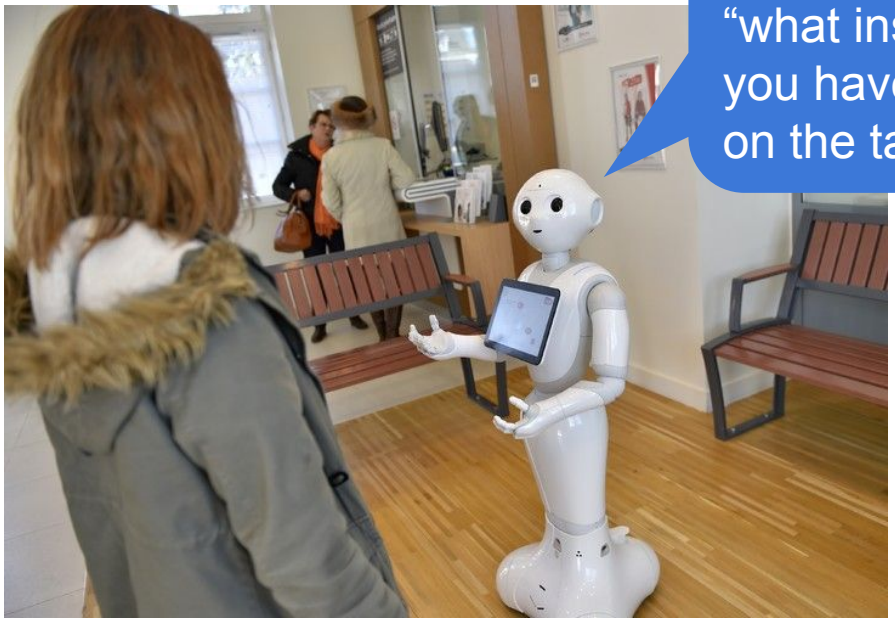


Theory:
What can I say to Pepper?

How can we help her know what to say?

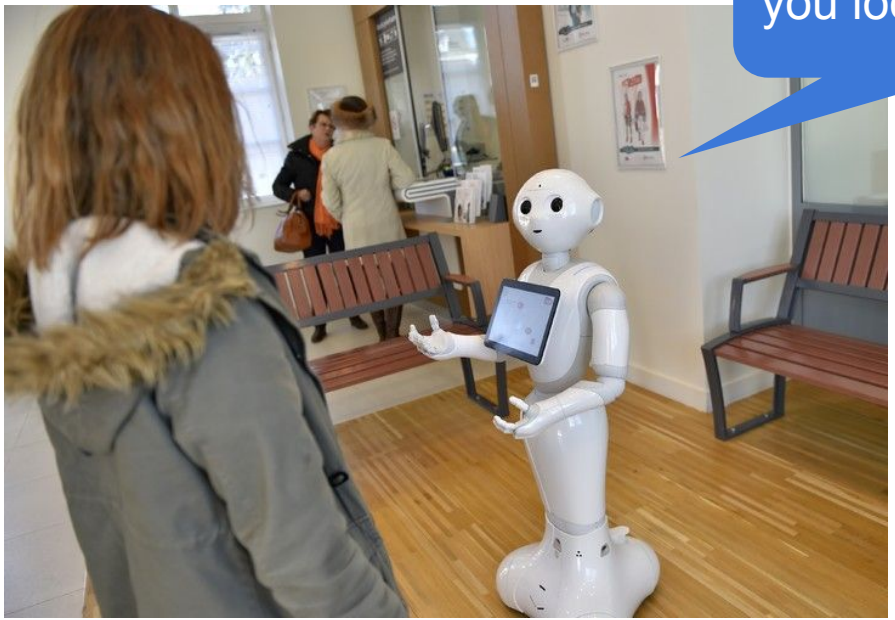


1) Just tell her!



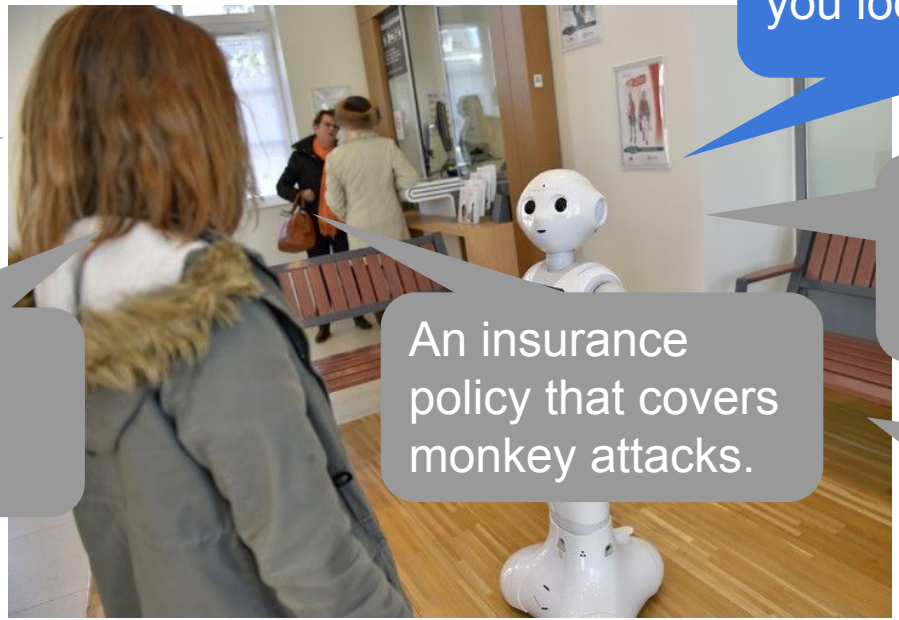
2) Show it on the tablet

3) Ask a natural question



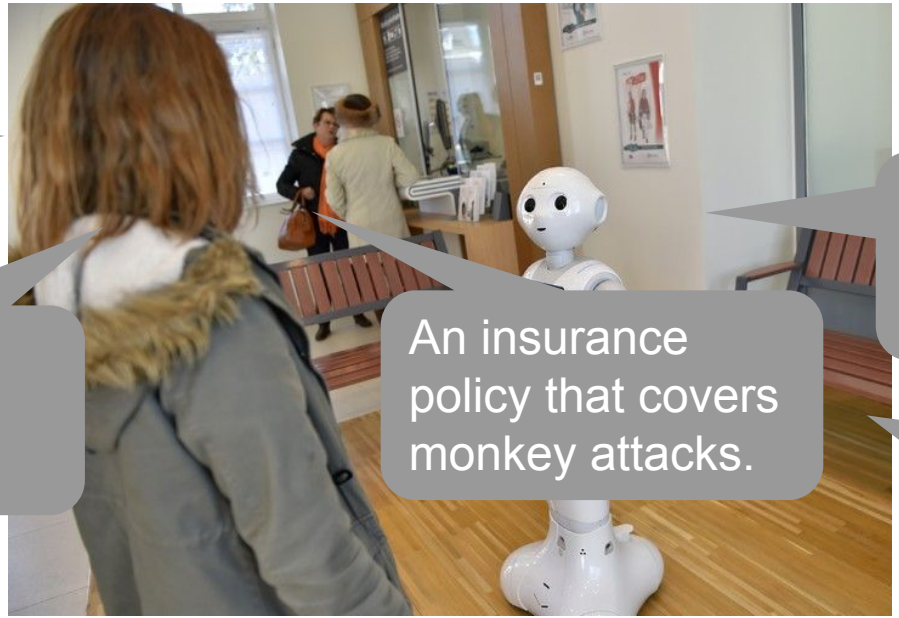
What can I say to Pepper?

(That isn't always enough)

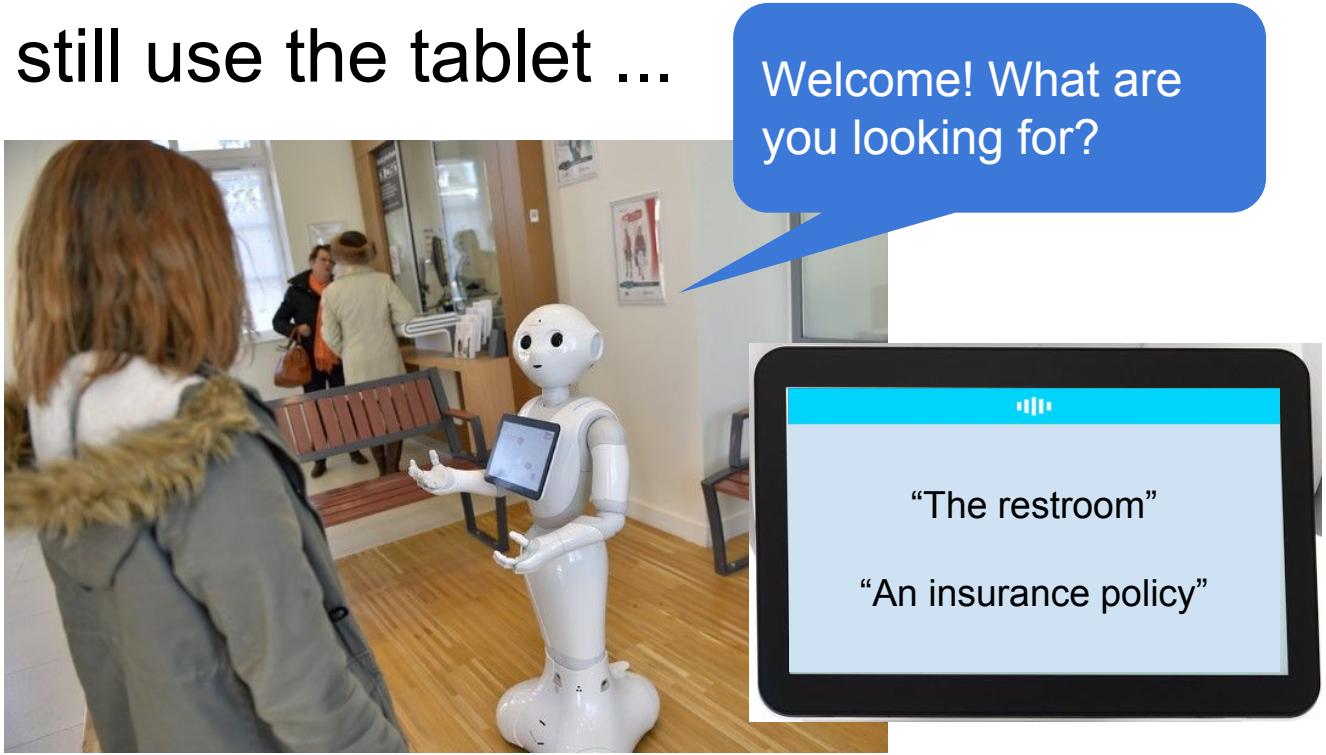


What can I say to Pepper?

… but write all that down, and add it to your QiChat!

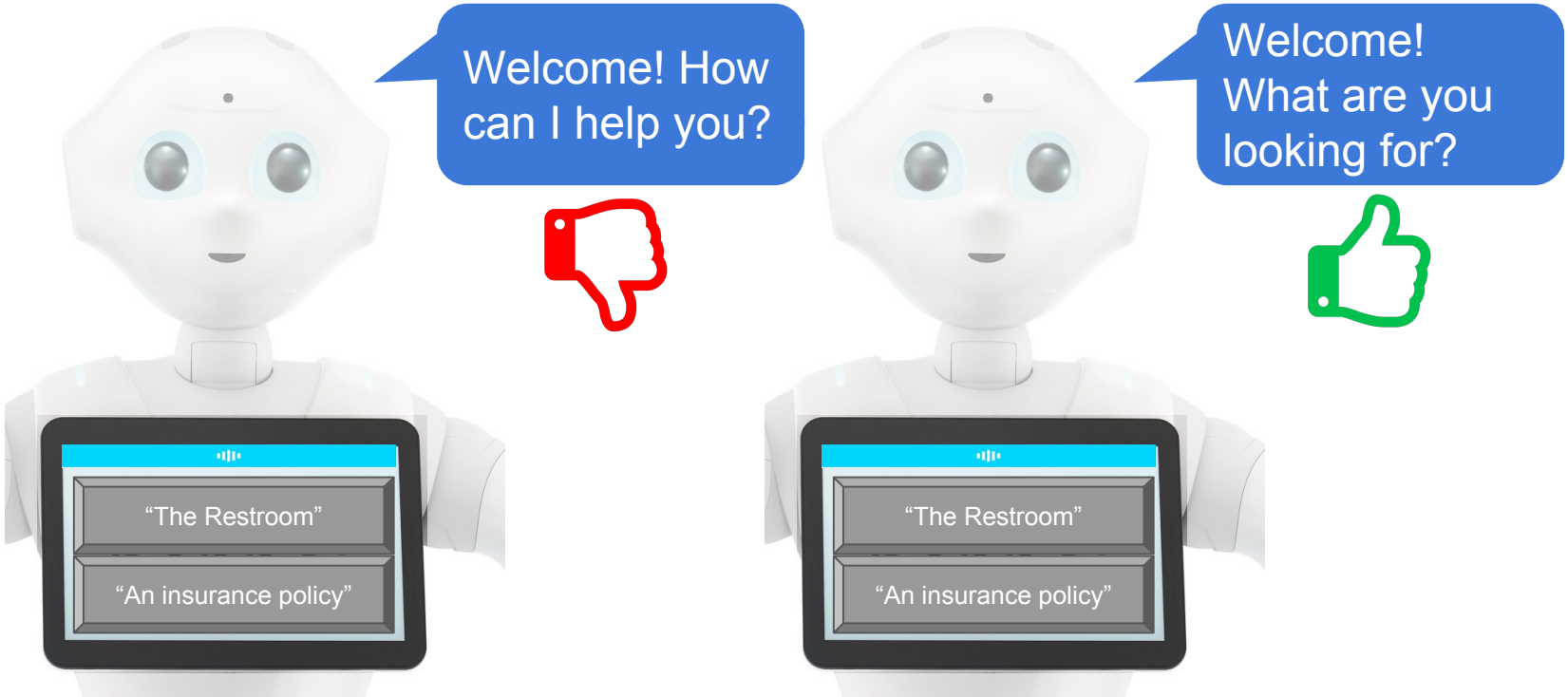


You can still use the tablet ...



What can I say to Pepper?

… but show answers that match the question !



What can I say to Pepper?

So to help people know what to say, use:

- Short and clear questions
- Short tablet suggestions
- A lot of testing with real people!

SoftBank Robotics

pepper

Hands-on:

Pepper, the car expert

Pepper, the car expert

A common use case is to have a database of products, and have Pepper be able to answer rich questions about these.

Let's do that with **cars**: Pepper's job is to show you the car you ask for.

Let's pretend we have a database of cars(in fact, it's a folder of images)

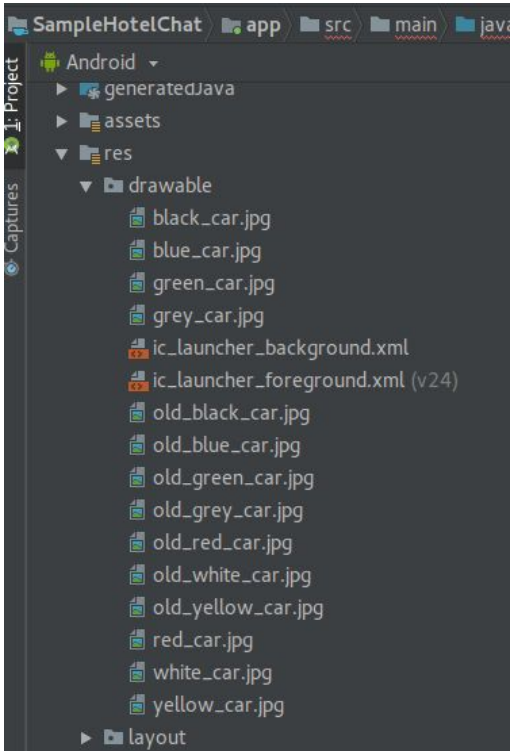
Pepper, the car expert

We will need:

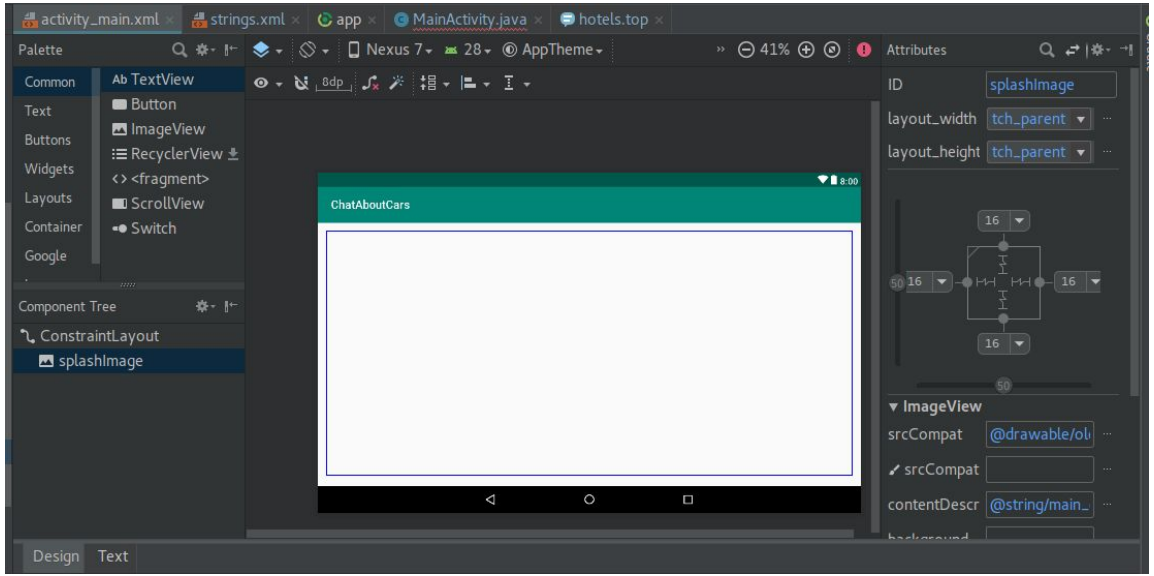
- To call code from qichat
- To show an image on the tablet
- To answer structured questions

Pepper, the car expert

Import the image folder,



...and add an **ImageView** to your layout.



Call it "`splashimage`", and make it invisible.

Pepper, the car expert

Let's start by creating an executor, to call java from QiChat.

In your topic:

```
proposal: %ICEBREAKER
  Ask me about cars!

u:(show me a car)
   here it is ^execute(showCar)
```

In your java (in `onRobotFocusGained`, after creating the chatbot):

```
HashMap<String, QiChatExecutor> executors = new HashMap<>();
executors.put("showCar", new ShowCarExecutor(qiContext));
chatbot.setExecutors(executors);
```

… we still need to create this `ShowCarExecutor` class

Pepper, the car expert

Create your executor class:

```
private class ShowCarExecutor extends BaseQiChatExecutor {
    public ShowCarExecutor(QiContext qiContext) {
        super(qiContext);
    }

    @Override
    public void runWith(List<String> params) {
        runOnUiThread(() -> {
            ImageView splashImage = findViewById(R.id.splashImage);
            splashImage.setImageResource(R.drawable.red_car);
            splashImage.setVisibility(View.VISIBLE);
        });
    }

    @Override
    public void stop() { }
}
```

Test your app: Pepper will now show you a car when asked to do so

Pepper, the car expert

That's nice, but Pepper always shows you the same car. Let's change that!

```
concept:(color) [black grey white red green blue yellow]

u:(show me a _~color [car one])
  Okay let's look for a $1 car ^execute(showCar, $1)
```

In your runWith(List<String> params):

```
if (params.size() > 0) {
   final String color = params.get(0);
   final String imageName = color + "_car";
   final int imageId = getResources().getIdentifier(imageName,"drawable",
           getPackageName());
   runOnUiThread(() -> {
       ImageView splashImage = findViewById(R.id.splashImage);
       splashImage.setImageResource(imageId);
       splashImage.setVisibility(View.VISIBLE);
   });
}
```

Pepper, the car expert

Let's polish our dialogue with concepts, and nested rules:

```
concept:(i_want) [
   "I want"
   "I'm looking for"
   "do you have"
   "show me"
   "can I see"
]
u:(~i_want a _~color [car one])
   Okay let's look for a $1 car ^execute(showCar, $1)

u:(~i_want a car)
   What color?
   u1:(_~color)
       Let's find a $1 car ^execute(showCar, $1)
```

SoftBank Robotics

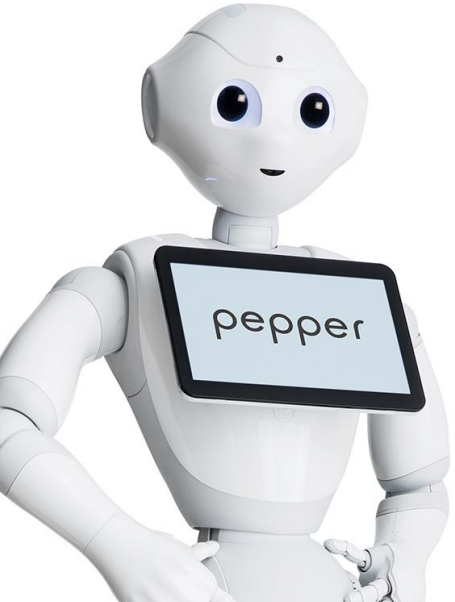
pepper

Challenge:

Everything about cars!

Challenge:

Make Pepper answer better questions about cars



Show me an old red car

What color?

Do you have vintage cars?

Okay, here's a blue one

Old, or new?

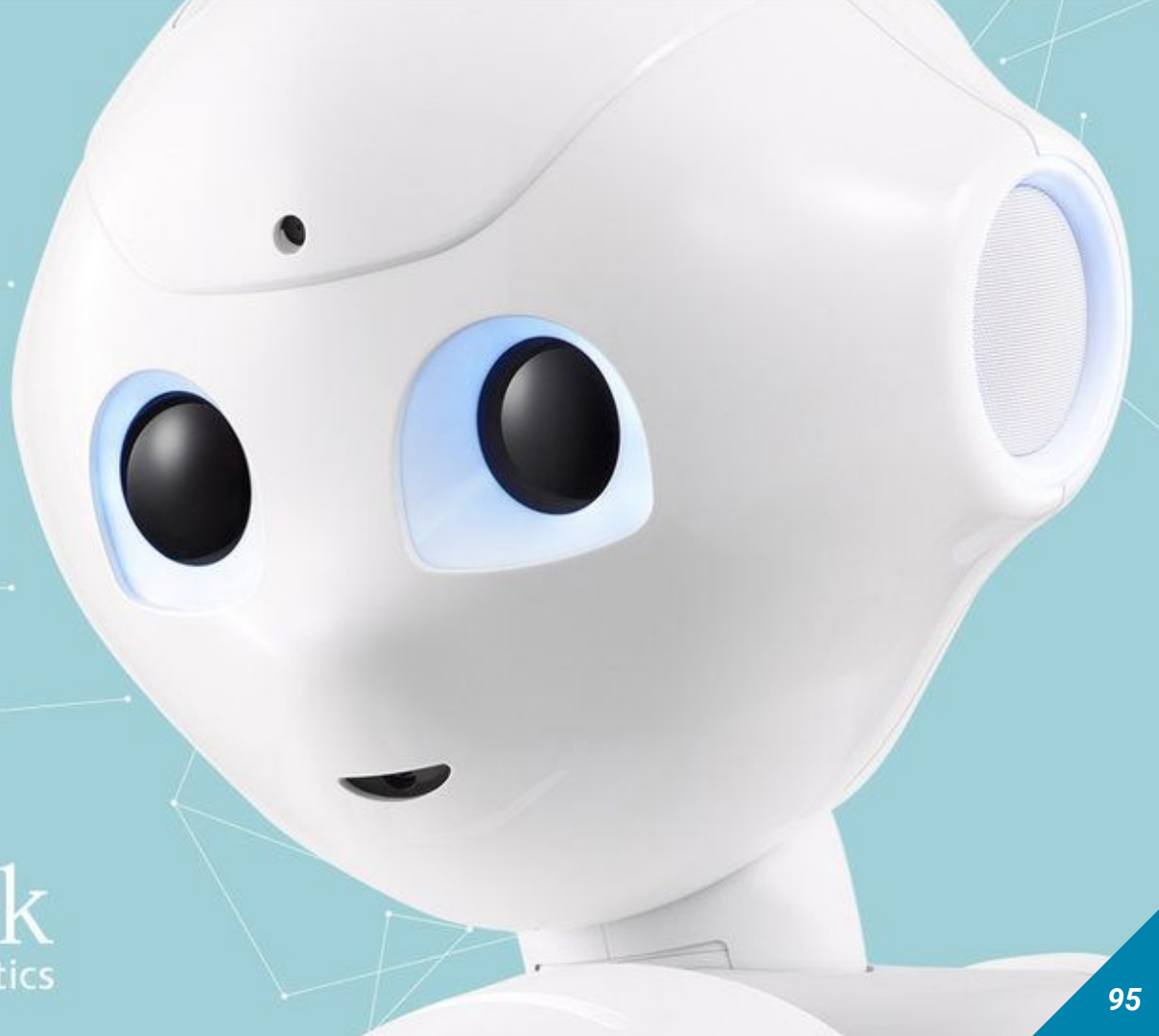
Show me a red car

Old

End of Day 1

See you tomorrow!

Day 2



SoftBank Robotics

Agenda for the next 3 days

Day 1

Discover Pepper

Discover The QiSDK

Conversation

Day 2

Enter Interaction

Asynchronous Programming

Navigation

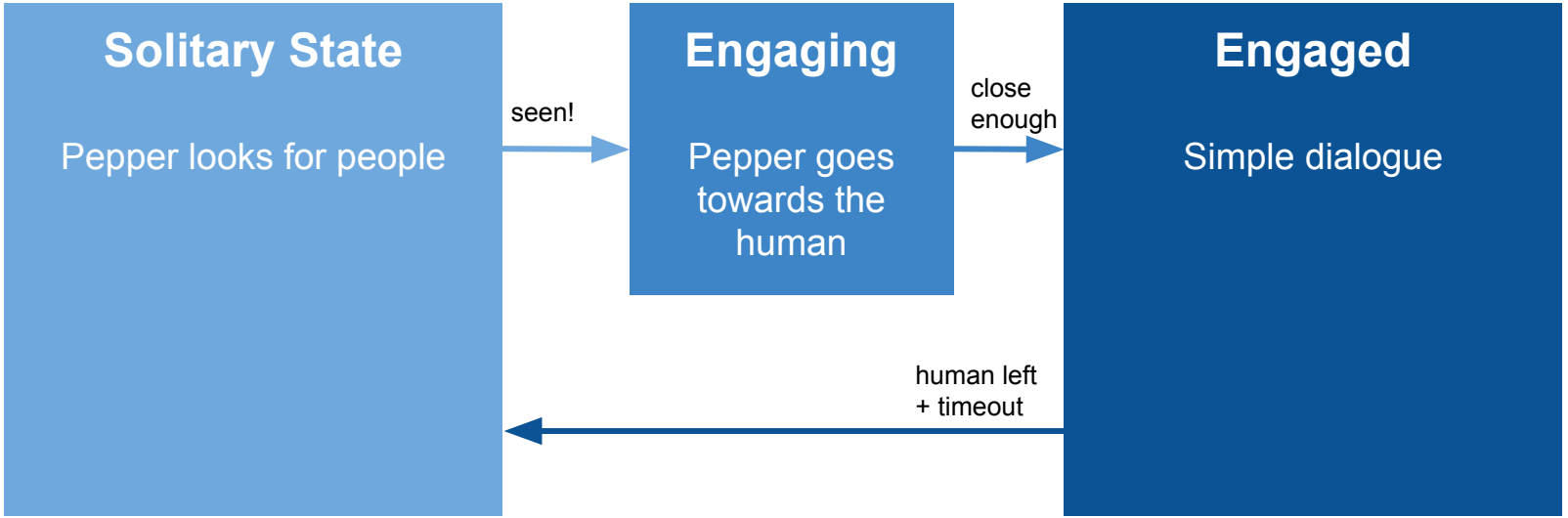
Day 3

Tips & Tricks

Making a full app

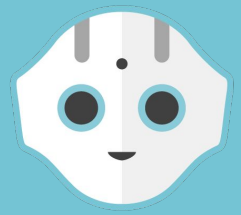
Where we're heading

We're heading towards making a more complete app flow:



SoftBank Robotics

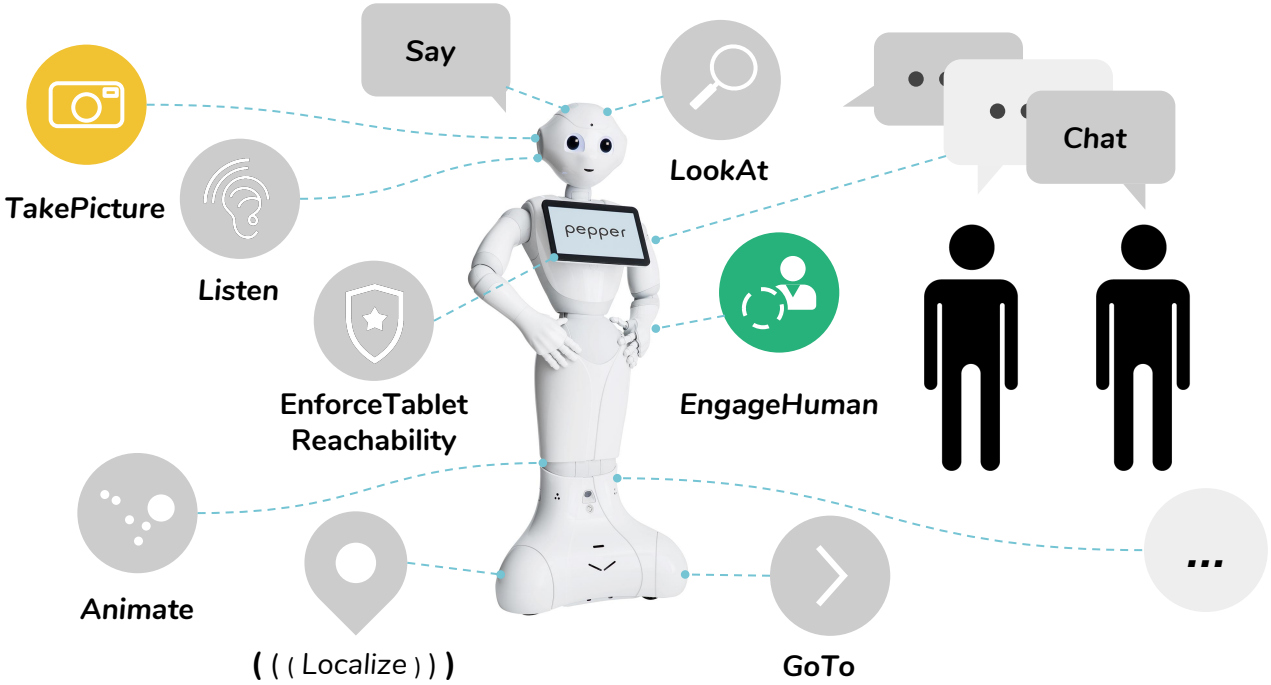
pepper



Theory:
Perception

QiSDK APIs: Perception

Let's look at [the documentation on Perception]



QiSDK APIs: Perception

Some important concepts in **Human Awareness**:

A **Human** object has useful characteristics:

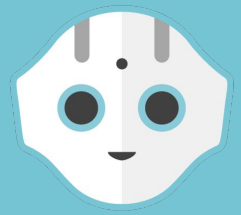
- Position in space (a **frame**)
- Face image (only when the image is good)
- Estimated age, gender, attention…

Several ways of accessing them:

- **Humans Around**: everyone Pepper can detect
- **Engaged Human**: the human (if any) Pepper is looking at
- **Recommended Human To Engage**: someone she *may* want to look at

SoftBank Robotics

pepper



Hands-on:
Enter Interaction

Reacting to people

Hi there !

Let's make Pepper work as an "announcer" - just say something to every person she sees.

For now, let's just use ***HumanAwareness****'s* ***onEngagedHumanChanged.***

Engage a Human

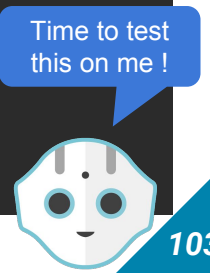
Step 1

- Create a new project

Step 2

- Add this code

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
   sayHello = SayBuilder.with(qiContext).withText("Hi there").build();
   humanAwareness = qiContext.getHumanAwareness();
   humanAwareness.addOnEngagedHumanChangedListener(this::onEngageHuman);
   onEngageHuman(humanAwareness.getEngagedHuman()); // init
}

public void onEngageHuman(Human engagedHuman) {
   if (engagedHuman!=null) {
       sayHello.run();
   }
}

@Override
public void onRobotFocusLost() {
   if (humanAwareness!=null) {
       humanAwareness.removeAllOnEngagedHumanChangedListeners();
   }
}
```



Reacting to people

What happens if Pepper sees someone else ? She might

- *Talk to him instead*
- *Or keep on talking to the first person*

Two tools can help you handle this:

- *The* **Recommended human to engage** *-> you get notified that there's a potential candidate*
- *The* **EngageHuman** *action -> allows you to control engagement.*

This way we can build a real **Interacting** state.

Human to Engage

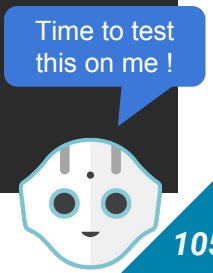
Step 1

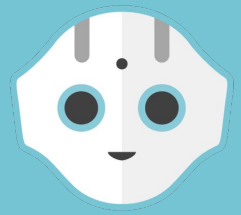
- Get the *recommended* human instead

Step 2

- Set a listener on the recommended human to engage

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
   this.qiContext = qiContext;
   sayHi = SayBuilder.with(qiContext).withText("Hi there").build();
   sayBye = SayBuilder.with(qiContext).withText("Bye!").build();
   humanAwareness = qiContext.getHumanAwareness();
   humanAwareness.addOnRecommendedHumanToEngageChangedListener(this::onRecommendedHuman);
   onRecommendedHuman(humanAwareness.getRecommendedHumanToEngage()); // init
}

public void onRecommendedHuman(Human human) {
   if ( (human!=null) && (state == STATE_SOLITARY)) {
       state = STATE_ENGAGED;
       EngageHuman engage = humanAwareness.makeEngageHuman(qiContext.getRobotContext(),
human);
       engage.addOnHumanIsEngagedListener(() -> sayHi.run());
       engage.addOnHumanIsDisengagingListener(() -> sayBye.run());
       engage.run();
       state = STATE_SOLITARY;
       onRecommendedHuman(nextHuman);
   } else {
       nextHuman = human; // Who do I talk to if this guy leaves?
   }
}
```





Async Programming

We will need asynchronous programming for:

- Running actions in parallel
- Running actions from callbacks in the UI thread
- Chaining actions
- Cancelling actions

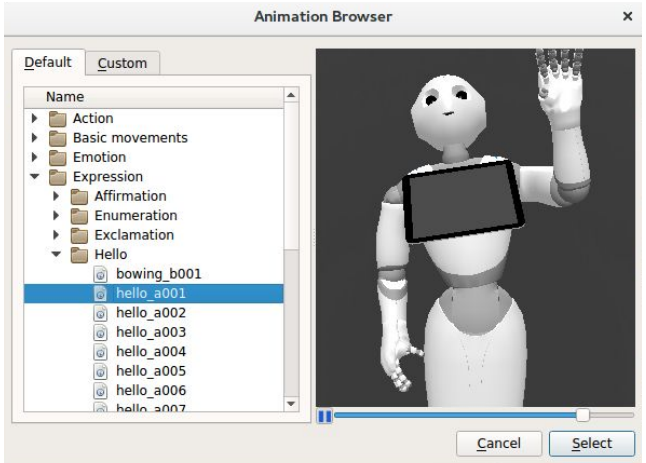
Adding animation

Let’s open our “hello world “ and add a second action: animate.

For this, we need to import an animation:

File -> New -> import animation

Note:
don’t confuse an “Animation” object and the “Animate” action.




```
@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withText("Hello everybody!")
            .build();
    Animation anim = AnimationBuilder.with(qiContext)
            .withResources(R.raw.hello_a001)
            .build();
    Animate animate = AnimateBuilder.with(qiContext)
            .withAnimation(anim)
            .build();
    say.run();
    animate.run();
}
```

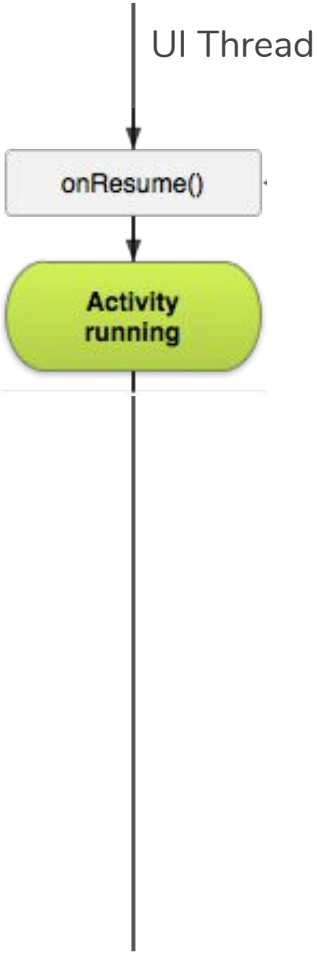
Time to test this on me !

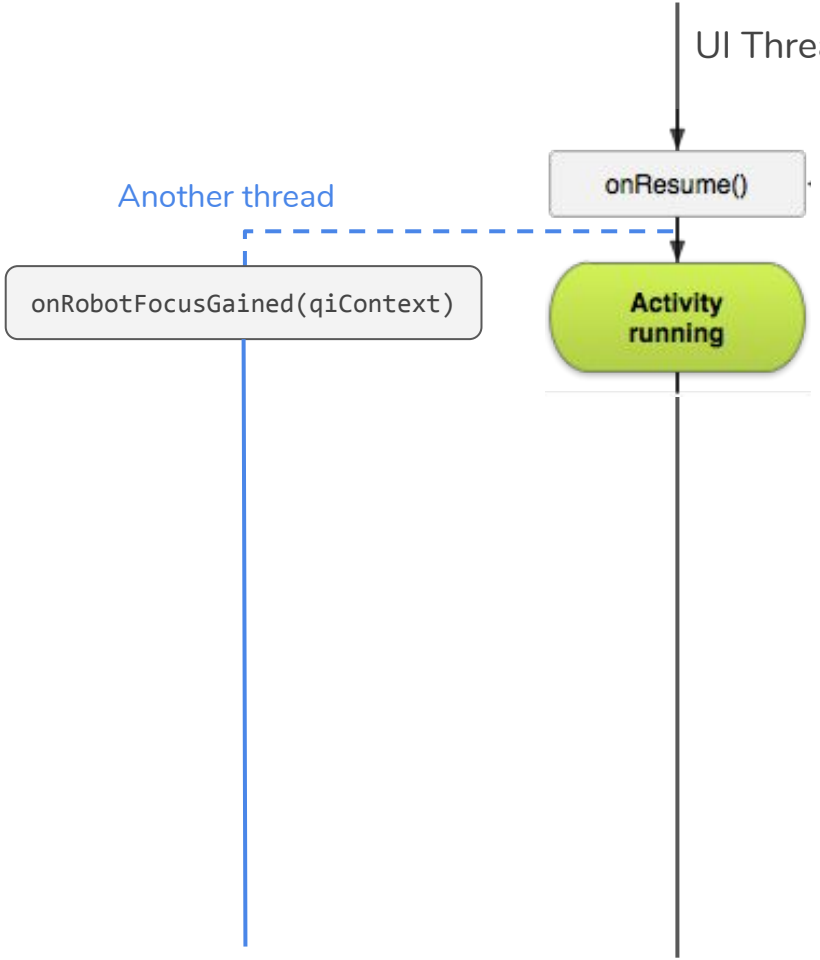
Adding animation

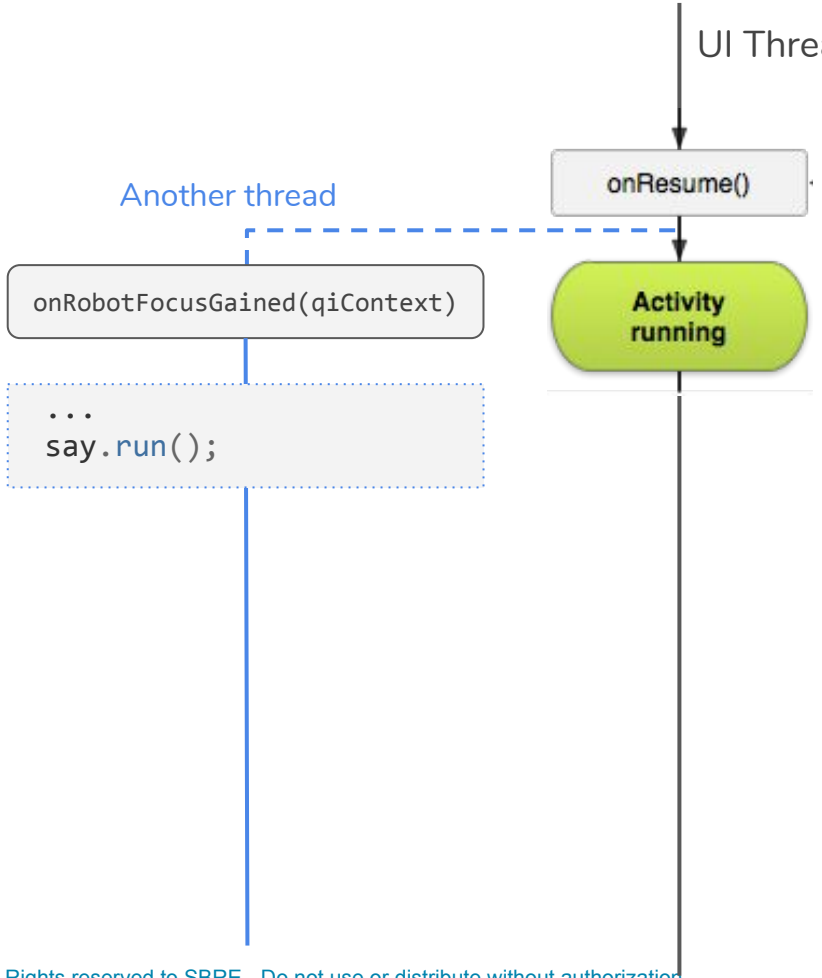
Not bad, but we want Pepper to speak and wave **at the same time**.

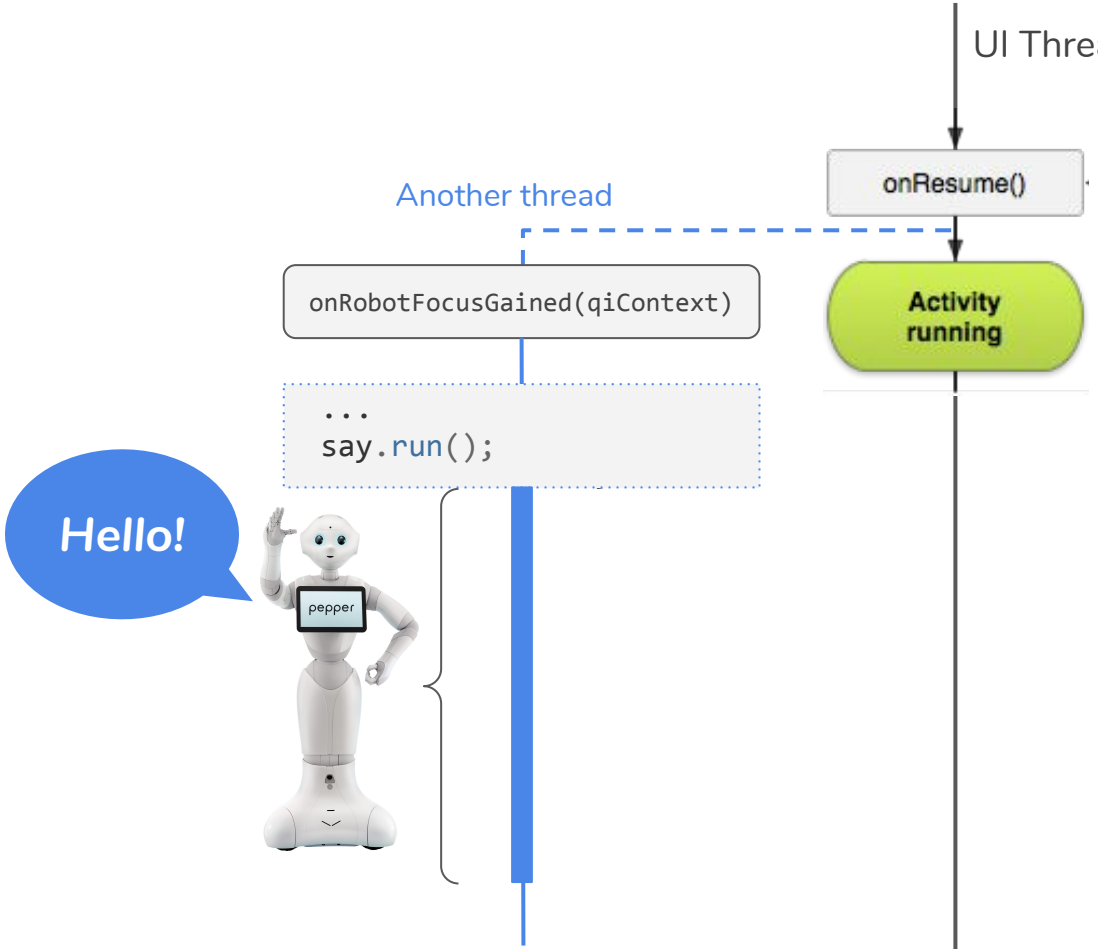
How? With ***async()***:

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
    Say say = SayBuilder.with(qiContext)
            .withText("Hello everybody!")
            .build();
    Animation anim = AnimationBuilder.with(qiContext)
            .withResources(R.raw.hello_a001)
            .build();
    Animate animate = AnimateBuilder.with(qiContext)
            .withAnimation(anim)
            .build();
    say.async().run();
    animate.run();
}
```

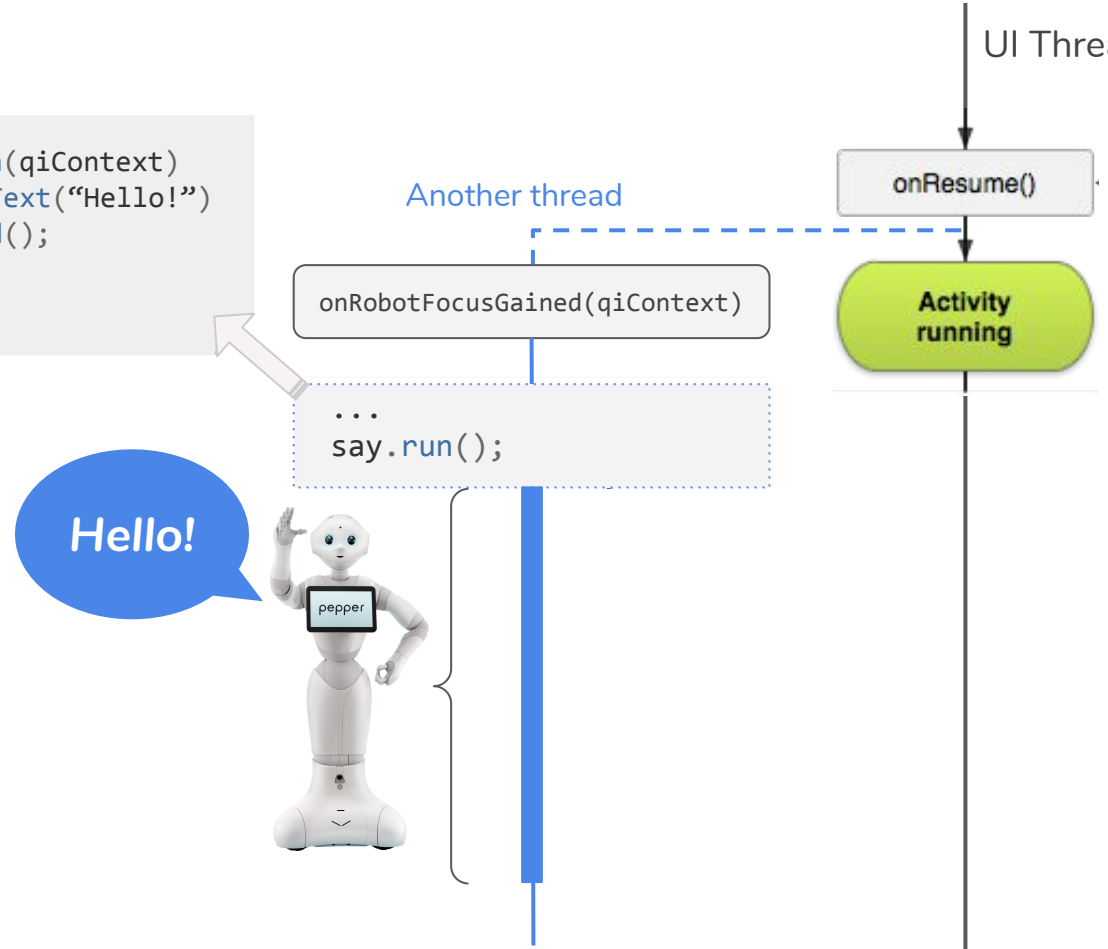




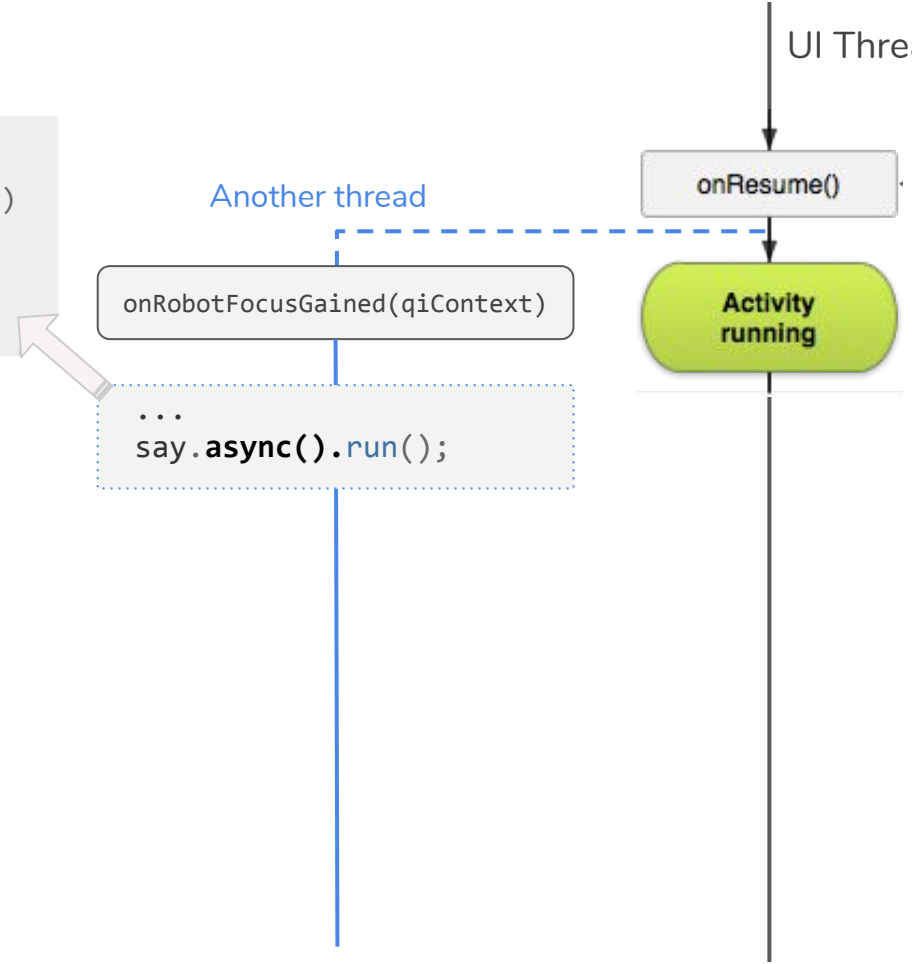




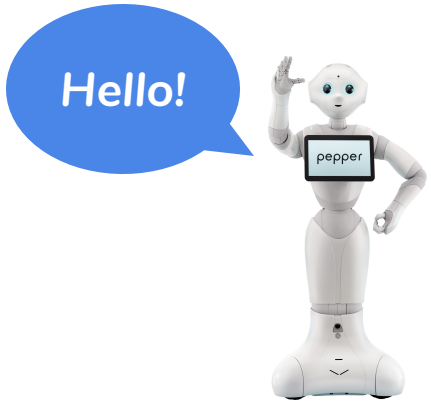
```
Say say = SayBuilder.with(qiContext)
                    .withText("Hello!")
                    .build();
say.async().run();
```

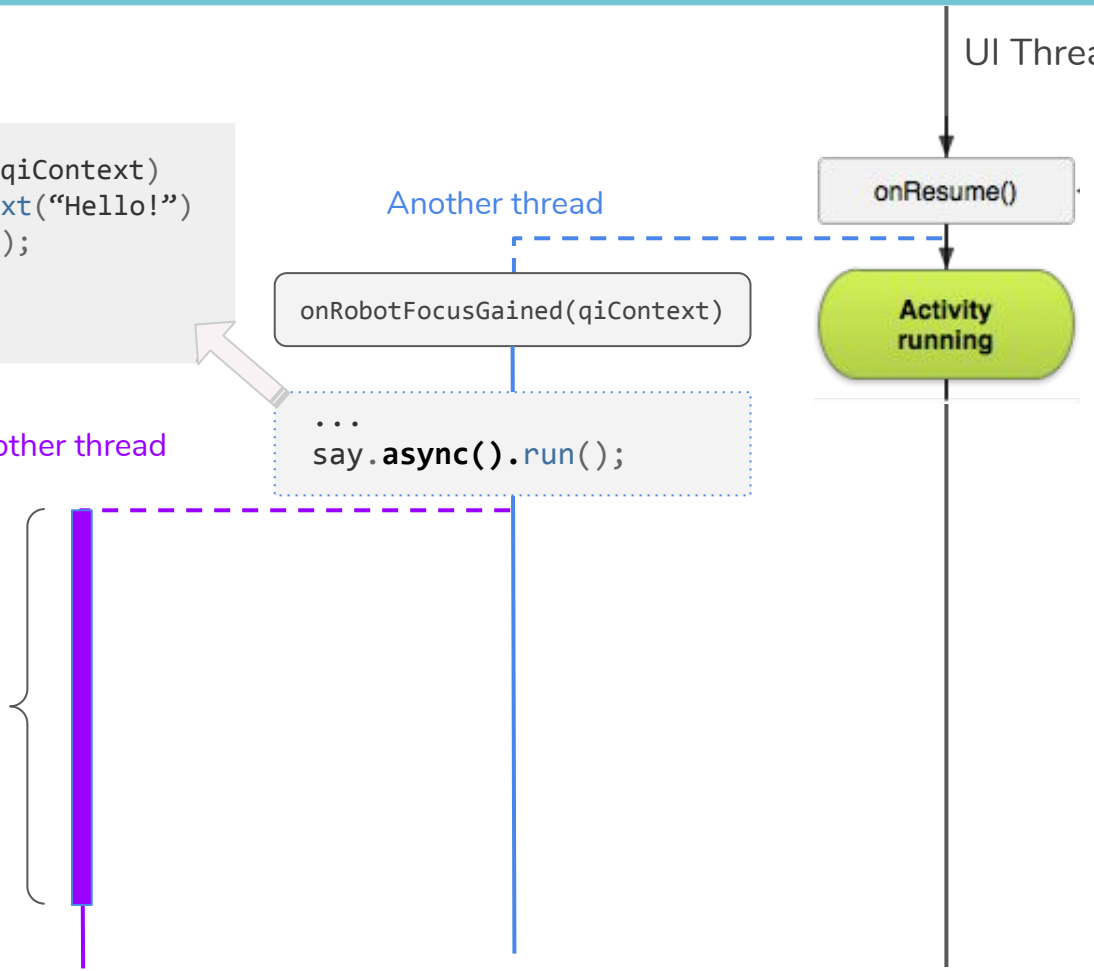


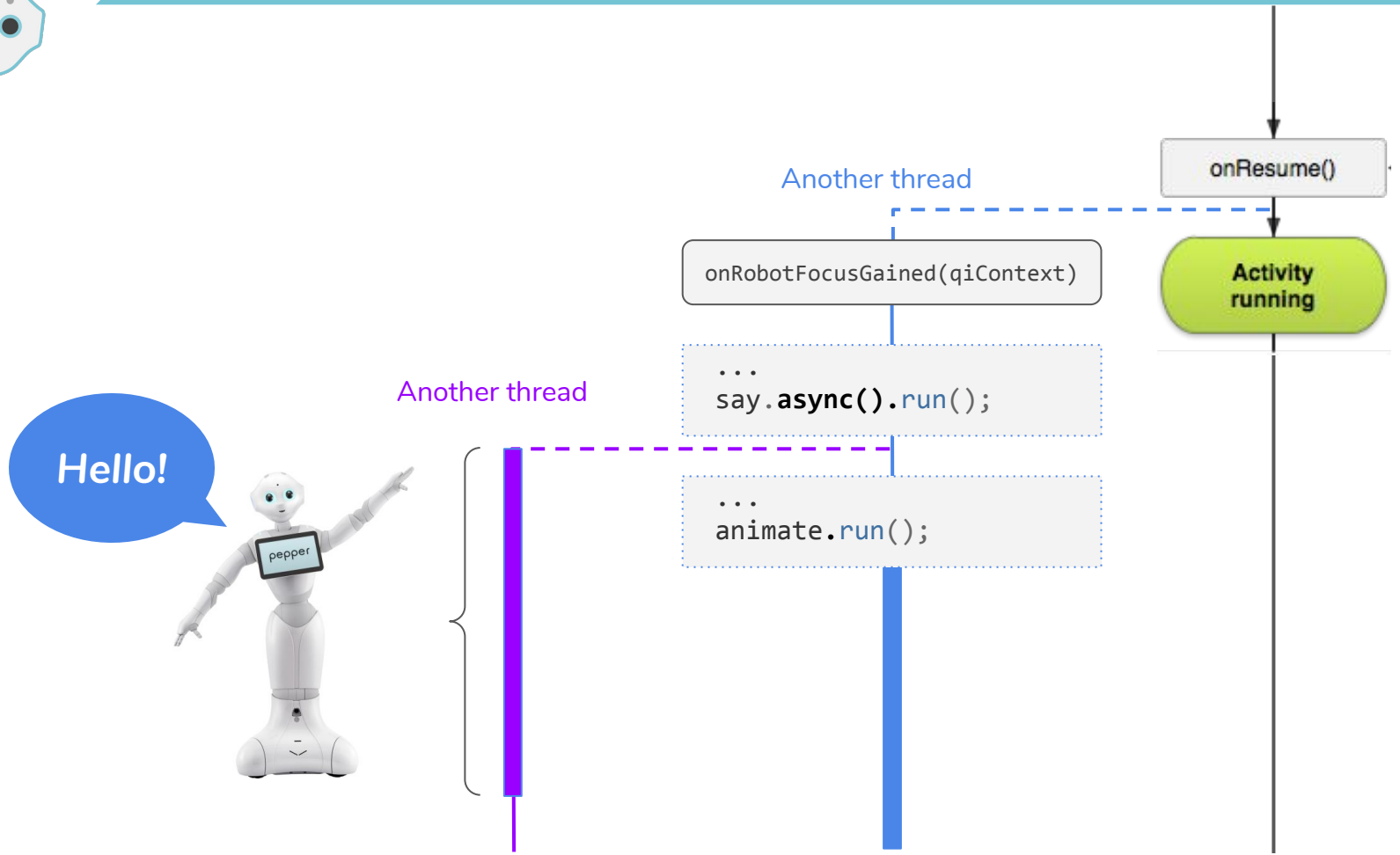
```
Say say = SayBuilder.with(qiContext)
                    .withText("Hello!")
                    .build();
say.async().run();
```



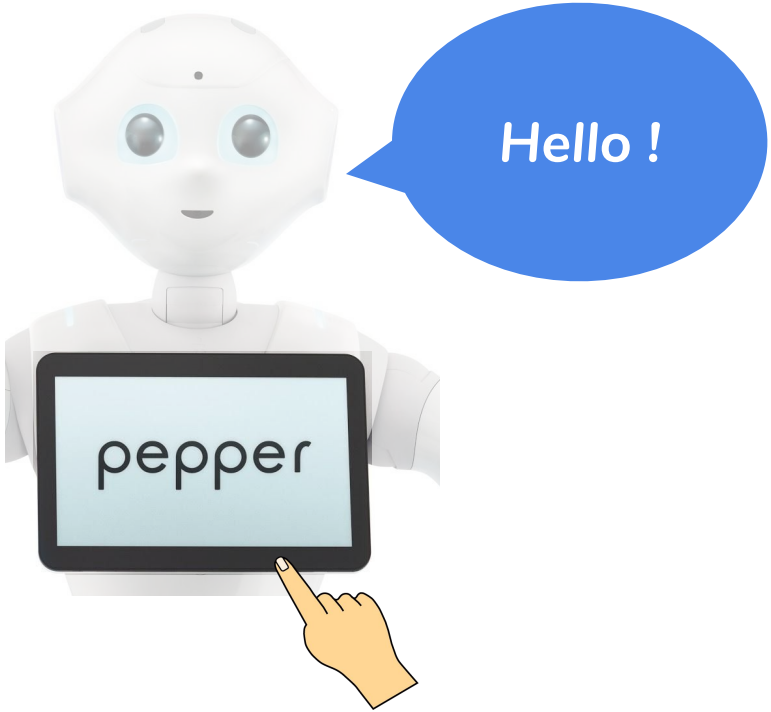
```
Say say = SayBuilder.with(qiContext)
                    .withText("Hello!")
                    .build();
say.async().run();
```







Now let’s make Pepper speak everytime we touch her tablet.



More on async

1. Add a button
2. Plug in a callback

This will not work!

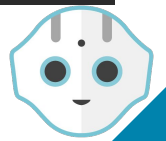
-> `NetworkOnMainThreadException`

Building and running the action requires network *(with the robot)* and must not be done on the UI Thread.

(also, what if qiContext is not ready?)

```
QiContext qiContext = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   QiSDK.register(this, this);

   final Button sayHelloButton = findViewById(R.id.sayhello);
   sayHelloButton.setOnClickListener((View v) -> {
       Say say = SayBuilder.with(qiContext)
               .withText("Hello everybody!")
               .build();
       say.run();
   });
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
   this.qiContext = qiContext;
}
```

Test this! (it won't work)



More on async

Two more changes are needed:

1. Build say outside UI thread
2. Run it with **async()**

```
QiContext qiContext = null;
Say say = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   QiSDK.register(this, this);

   final Button sayHelloButton =
findViewById(R.id.sayhello);
   sayHelloButton.setOnClickListener((View v) -> {
       if (say != null) {
           say.async().run();
       }
   });
}

@Override
public void onRobotFocusGained(QiContext qiContext) {
   this.qiContext = qiContext;
   say = SayBuilder.with(qiContext)
           .withText("Hello everybody!")
           .build();
}
```

What does async() do exactly ?

```
say.run();
```

This is a blocking call, that will continue when Pepper is done speaking.

It cannot be done on UI thread.

```
Future<Void> sayFuture = say.async().run();
```

This call immediately returns a **Future** object, representing the execution.

More on async

You can get notified of th
future's state.

... but this will crash !

Can you find the error?

```
java.lang.RuntimeException:
Can't create handler inside
thread that has not called
Looper.prepare()
```

```java
private void showToast(final String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}

private void onSayDone(Future<Void> _) {
    if (sayFuture.hasError()) {
        showToast("Error: " + sayFuture.getErrorMessage());
    } else {
        showToast("Say Done");
    }
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);

    final Button sayHelloButton = findViewById(R.id.sayhello);
    sayHelloButton.setOnClickListener((View v) -> {
        if (say != null) {
            Future<Void> sayFuture = say.async().run();
            sayFuture.thenConsume(this::onSayDone);
        }
    });
}
```

More on async

For the toast, you need runOnUiThread

(as for many other UI operations)

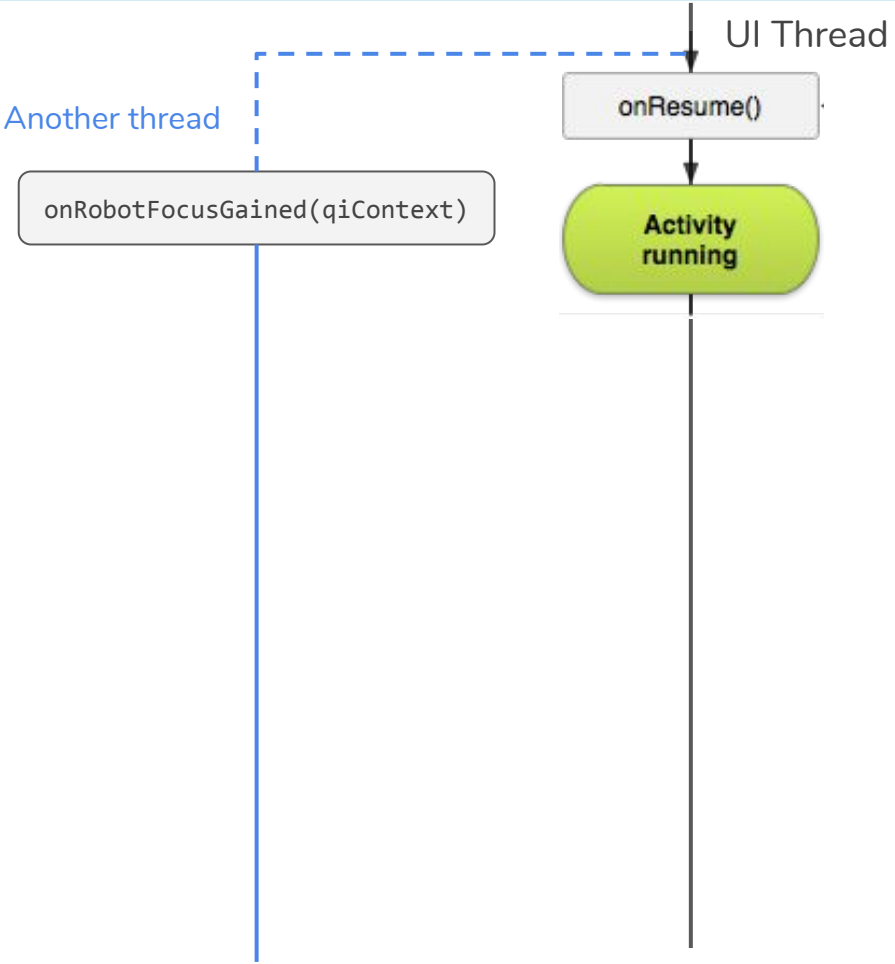
```
private void showToast(final String msg) {
    runOnUiThread(() -> {
        Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
    });
}

private void onSayDone(Future<Void> _) {
    if (sayFuture.hasError()) {
        showToast("Error: " + sayFuture.getErrorMessage());
    } else {
        showToast("Say Done");
    }
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);

    final Button sayHelloButton = findViewById(R.id.sayhello);
    sayHelloButton.setOnClickListener((View v) -> {
        if (say != null) {
            Future<Void> sayFuture = say.async().run();
            sayFuture.thenConsume(this::onSayDone);
        }
    });
}
```

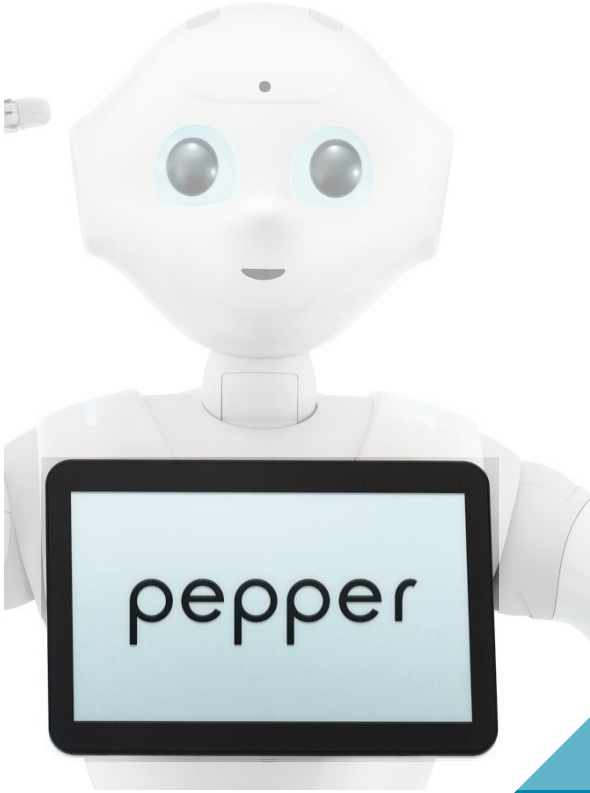
More on async

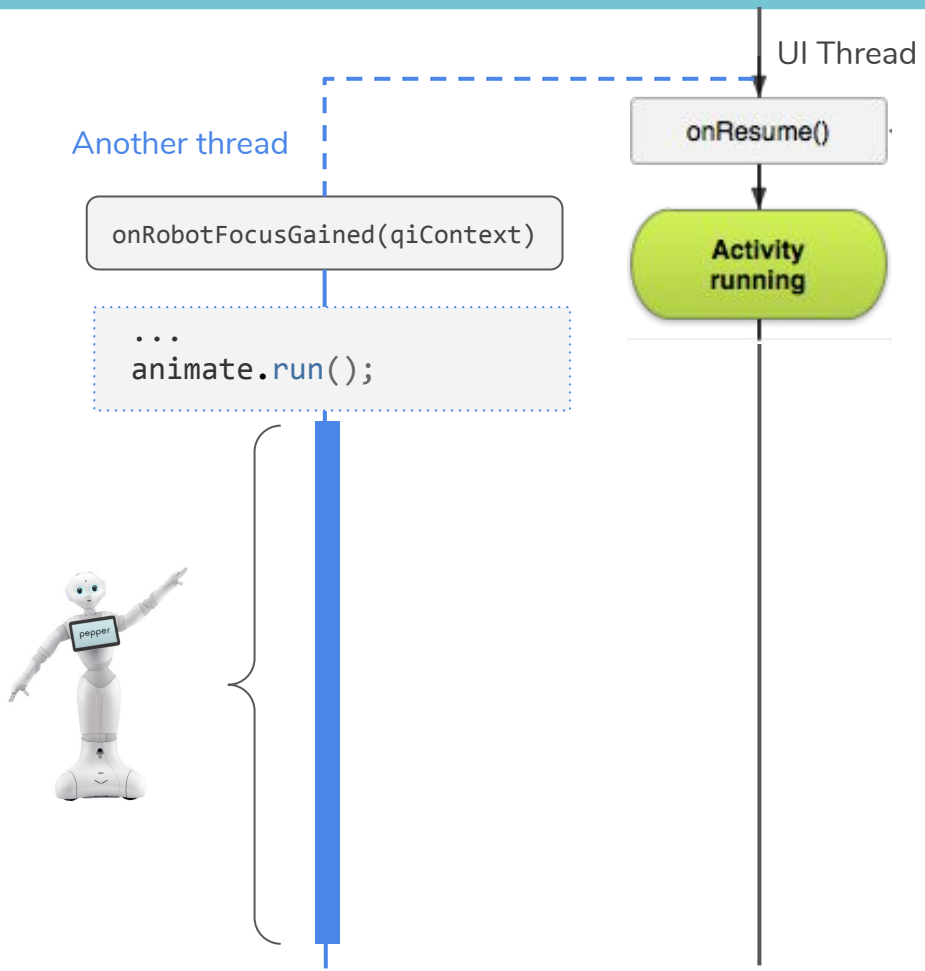
You can also **cancel** futures:

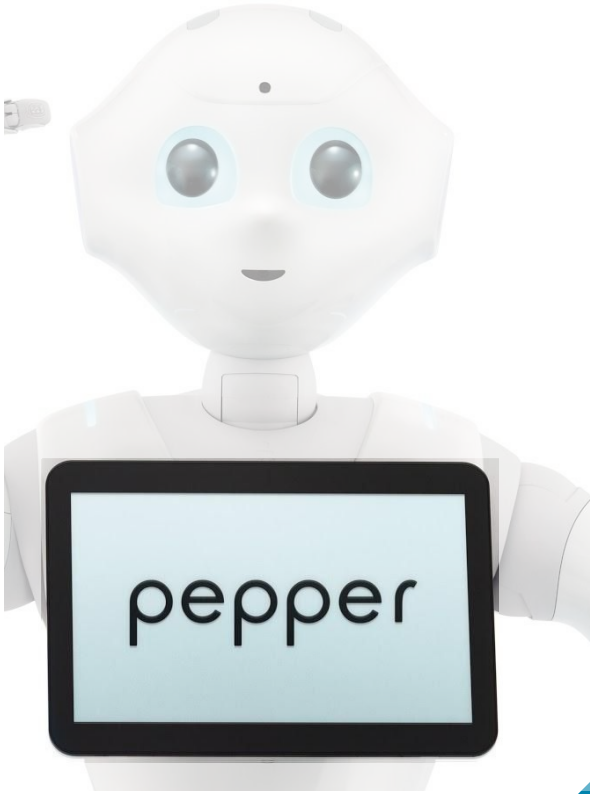
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    QiSDK.register(this, this);

    final Button sayHelloButton = findViewById(R.id.sayhello);
    sayHelloButton.setOnClickListener((View v) -> {
        if (say == null) {
        } else if ((sayFuture == null) || (sayFuture.isDone())) {
            sayFuture = say.async().run();
            sayFuture.thenConsume(this::showFutureResult);
        }
        else {
            // Say future is not done: cancel it
            sayFuture.requestCancellation();
        }
    });
}
```

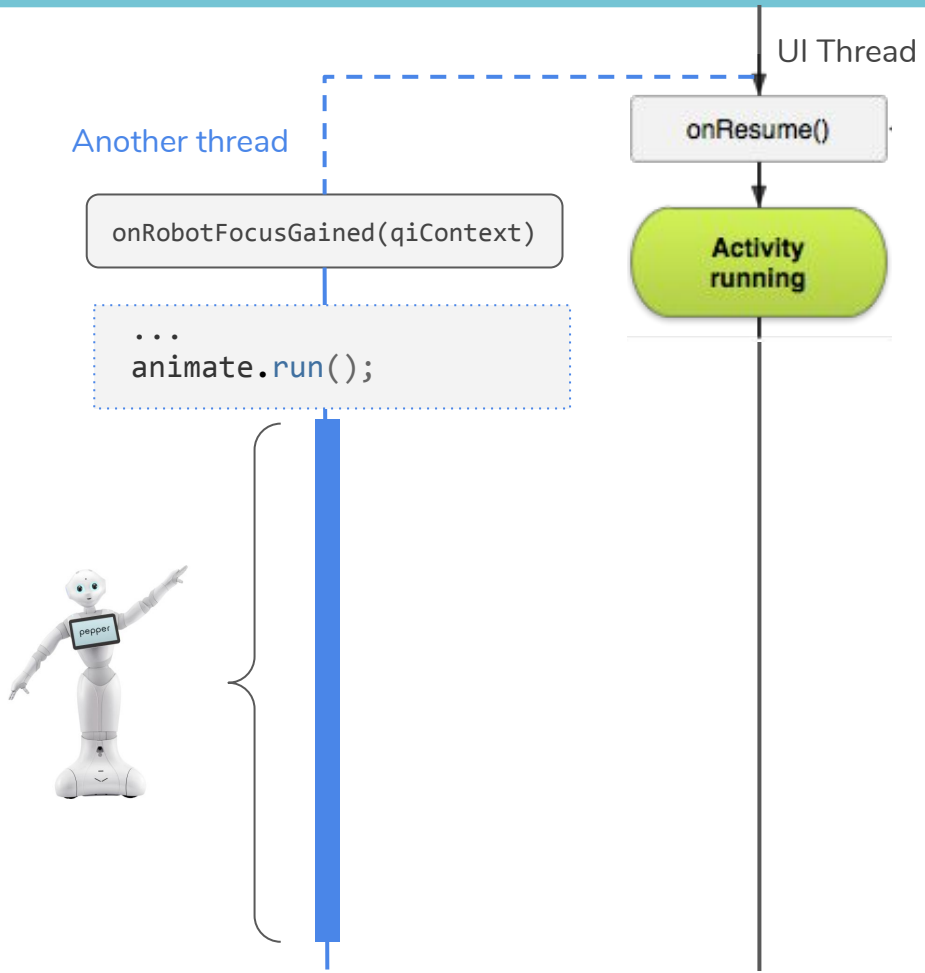
Running an action synchronously and trying to cancel

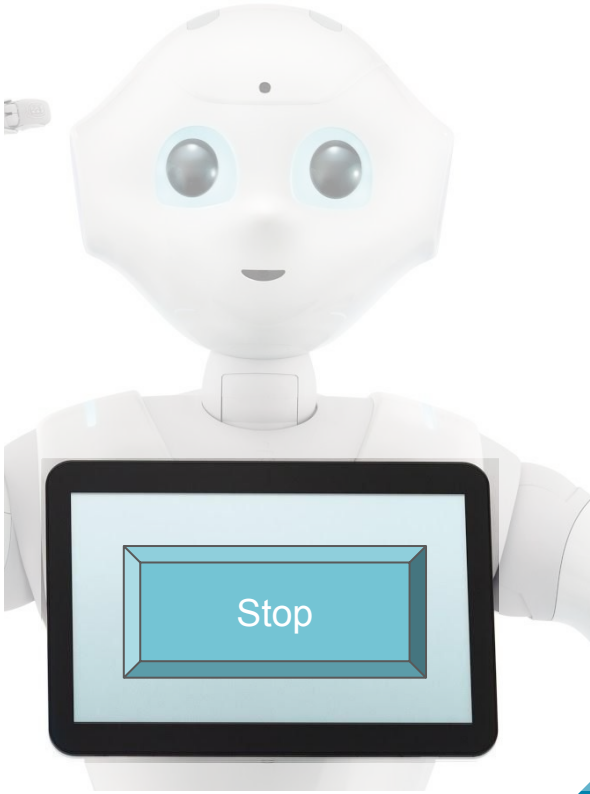


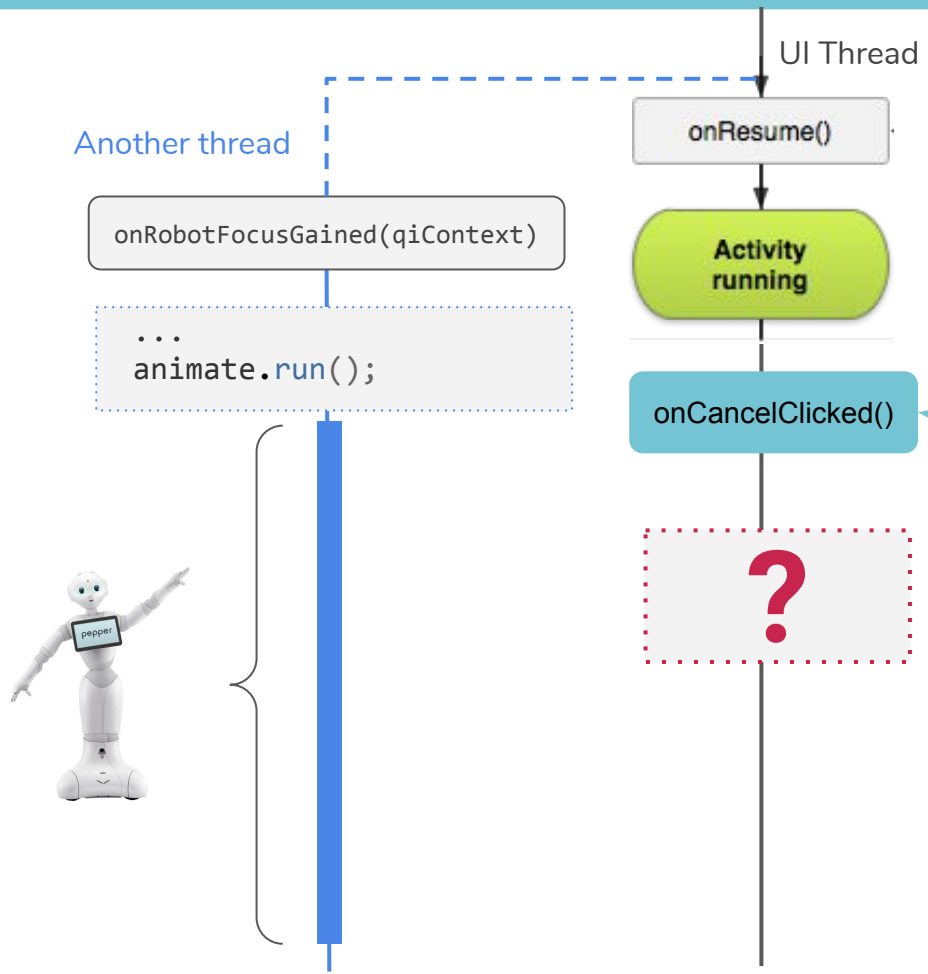


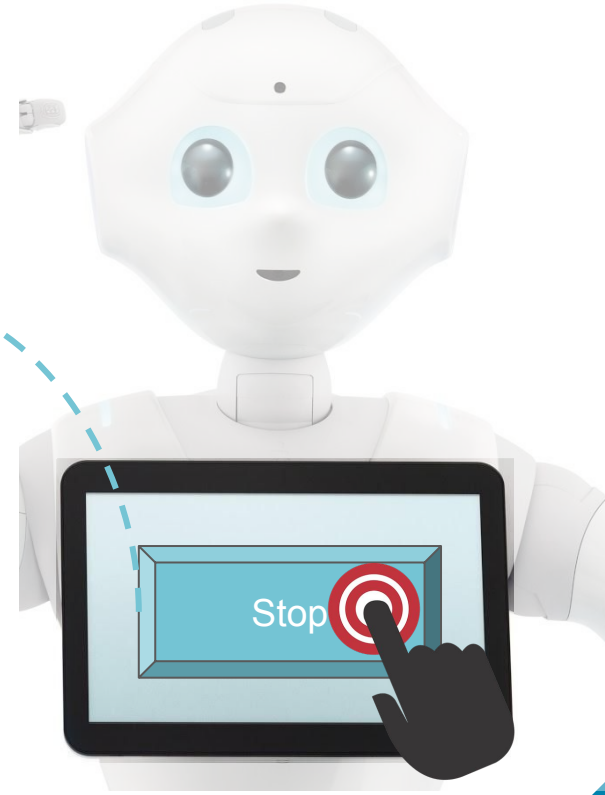


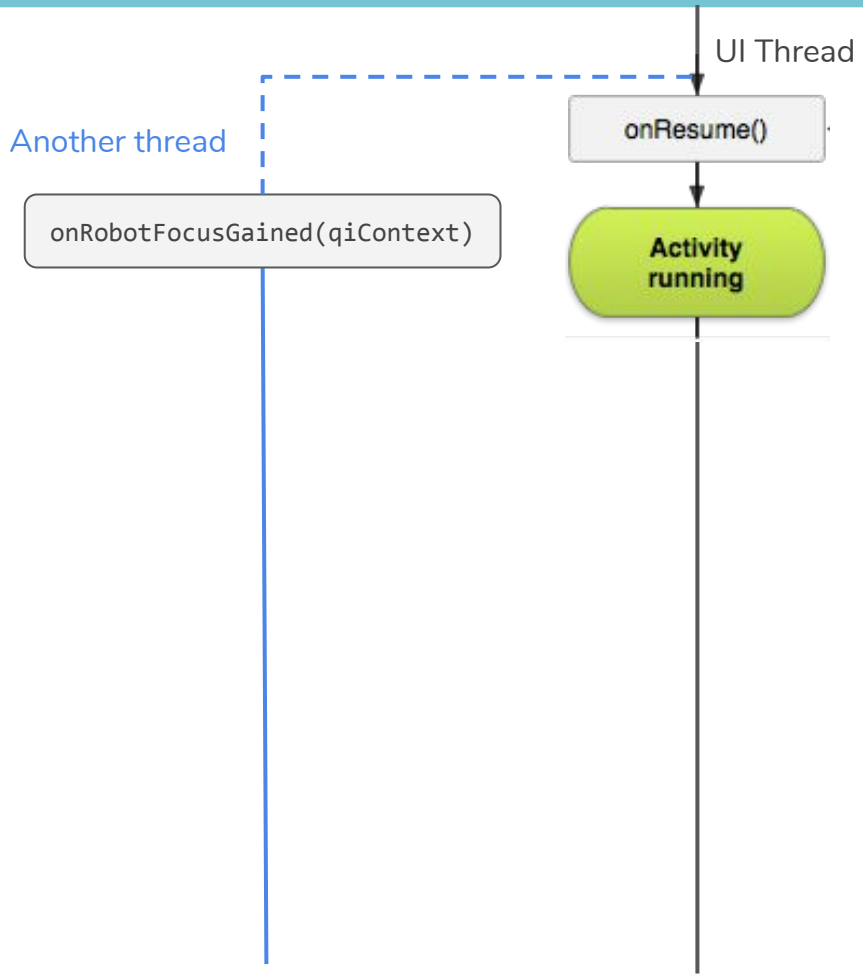


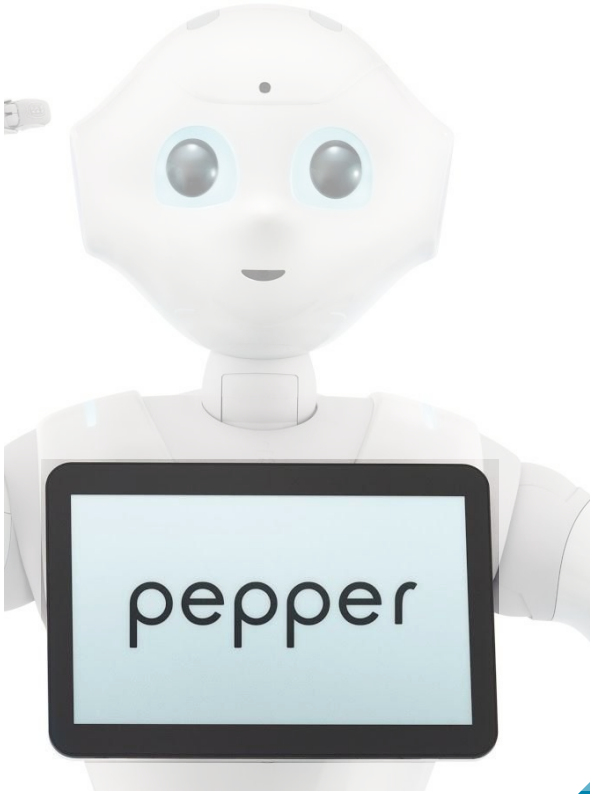


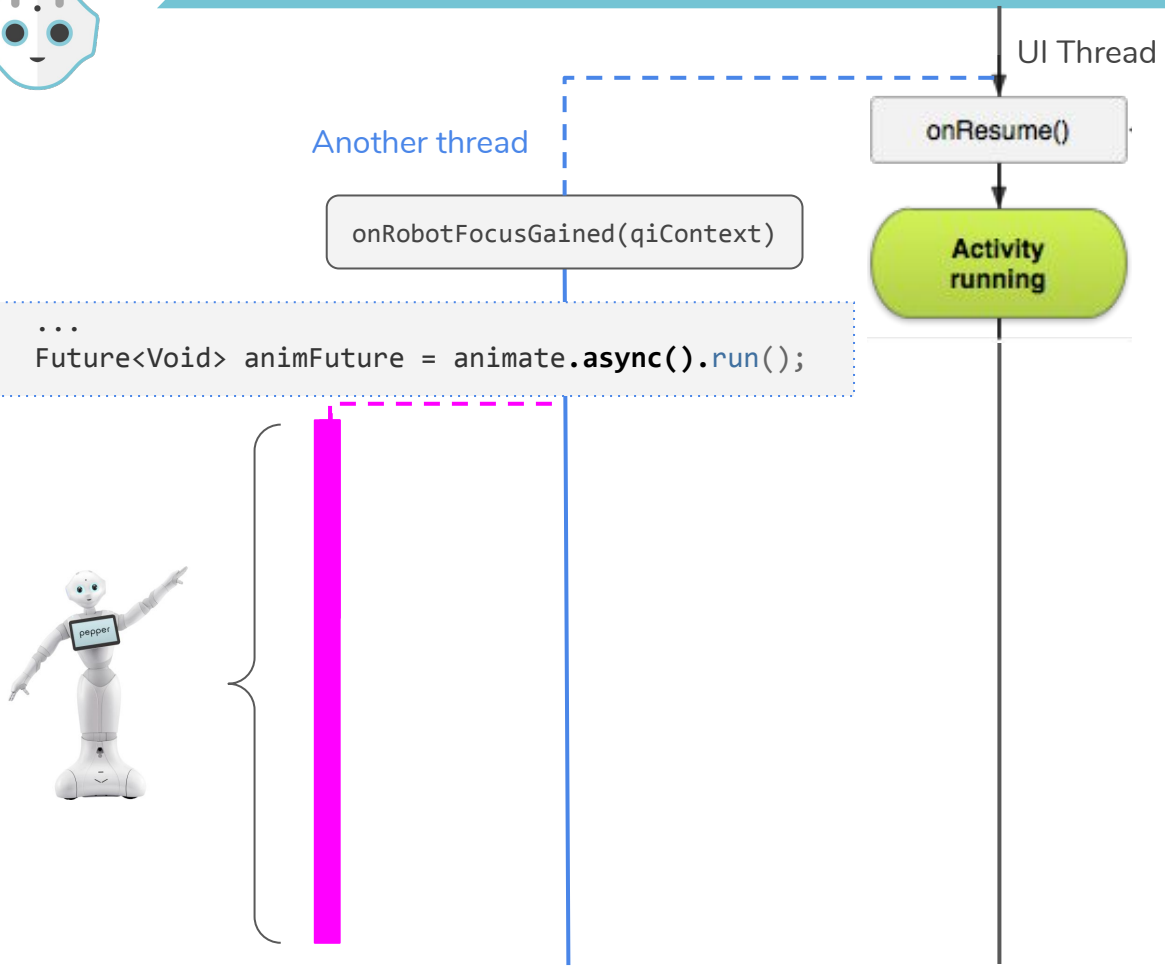


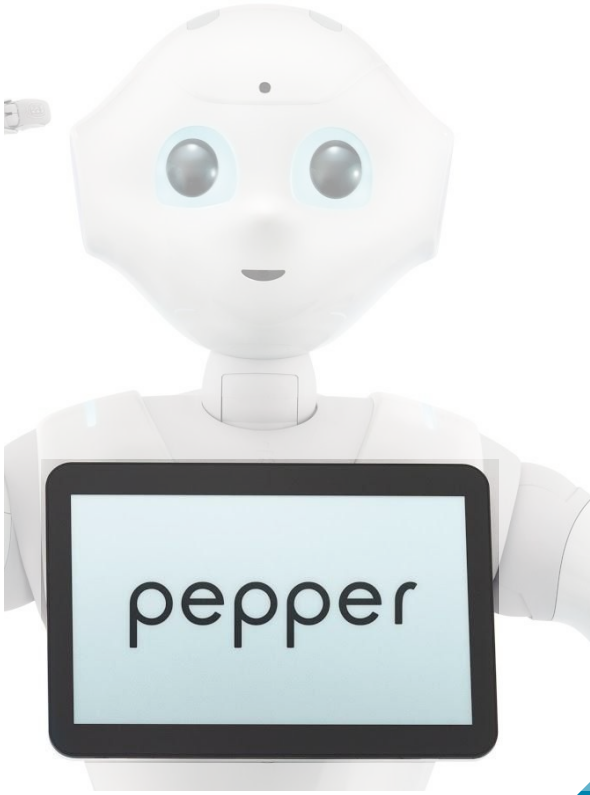


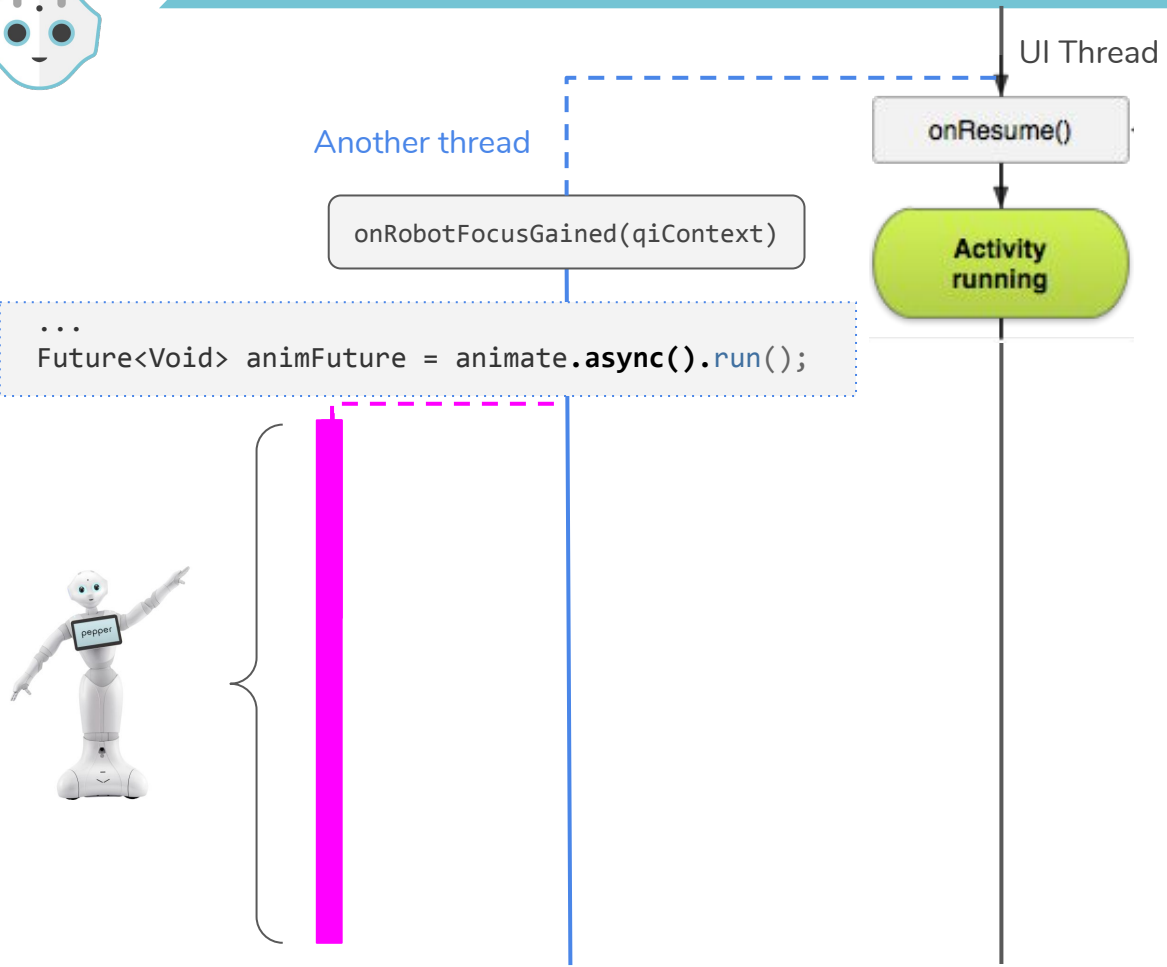


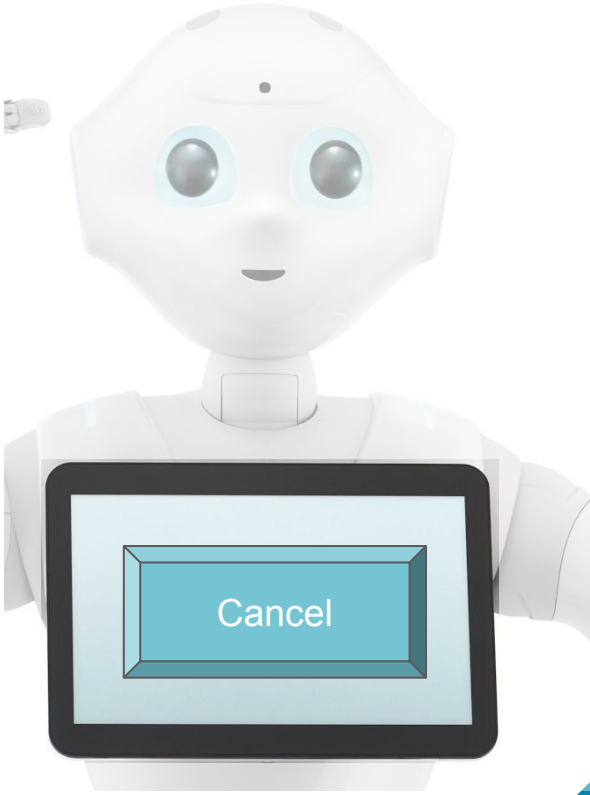












UI Thread

onResume()

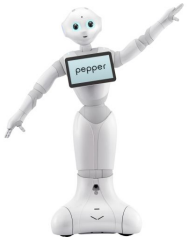
Activity running

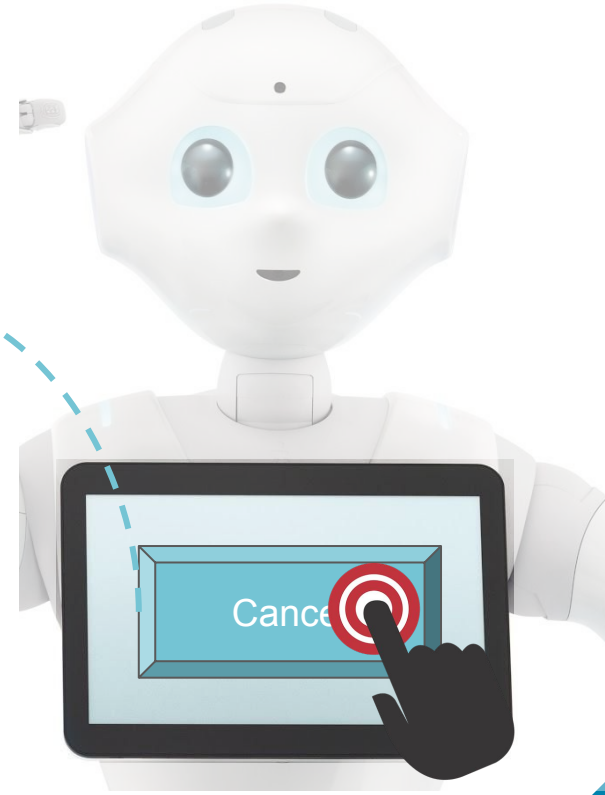
onCancelClicked()

Another thread

onRobotFocusGained(qiContext)

```
...
Future<Void> animFuture = animate.async().run();
```





UI Thread

onResume()

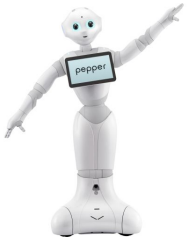
Activity running

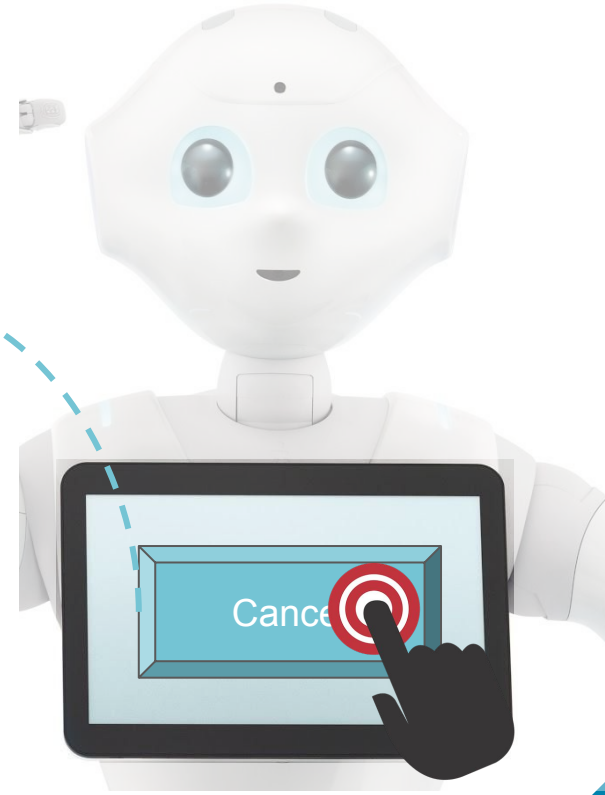
onCancelClicked()

Another thread

onRobotFocusGained(qiContext)

```
...
Future<Void> animFuture = animate.async().run();
```





UI Thread

onResume()

Activity running

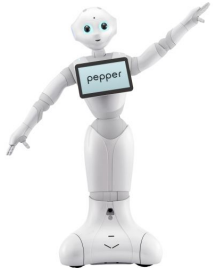
Another thread

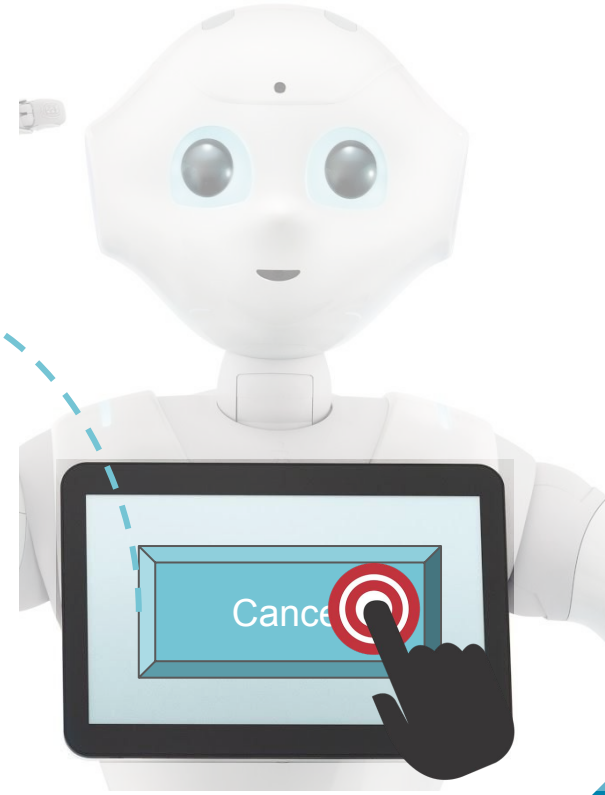
onRobotFocusGained(qiContext)

```
...
Future<Void> animFuture = animate.async().run();
```

onCancelClicked()

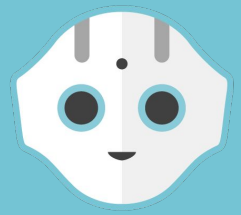
```
...
animFuture.requestCancellation();
```





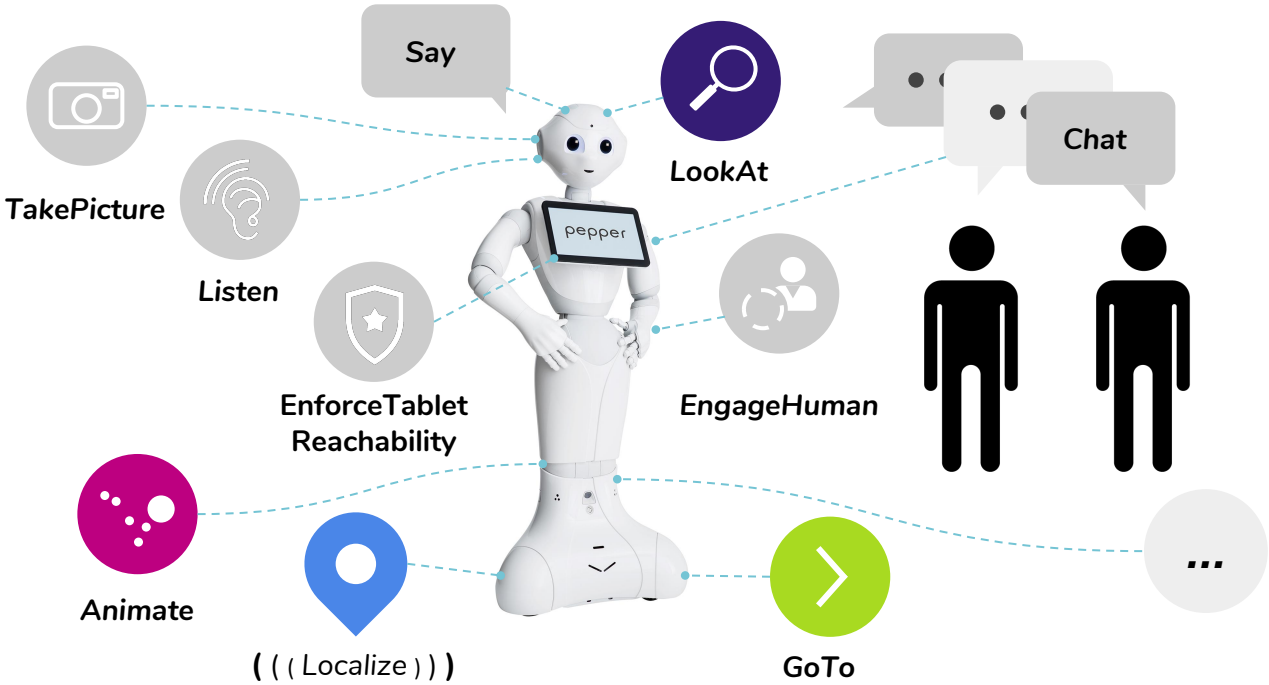
To recap, **.async()** is useful for:

- Running actions in parallel
- Running actions from callbacks in the UI thread
- Chaining actions
- Cancelling actions



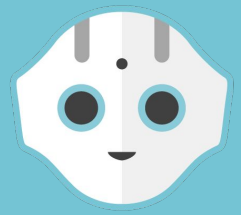
Theory:
Motion

Let's look at [the documentation on Motion]



SoftBank Robotics

pepper



Hands-on:
Navigation

Navigation

Pepper has some useful APIs for controlling Motion and Navigation which can be combined for a variety of use cases:

- Make Pepper navigate throughout his environment to appear dynamic
- Make Pepper actively move towards a human to interact, and then return to his base
- Use Pepper to patrol an area

We will focus on two concepts:

Frames - *Represents a spatial position*

GoTo - *Make Pepper navigate to a location*

Building the Frame

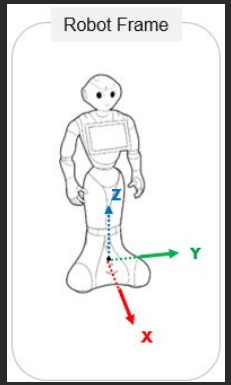
Step 1

- Create a new project
- Robotify the project

Step 2

- Get the current robot Frame to use as a reference
- Build a Target Frame 1 meter in front

```
@Override
public void onRobotFocusGained(QiContext qiContext)
{
   //Get the Robot Frame
   Actuation actuation = qiContext.getActuation();
   Frame robotFrame = actuation.robotFrame();

   //Move 1 Meter Forward (X Axis is forward/backward)
   Transform transform = TransformBuilder.create()
                  .fromXTranslation(1);

   Mapping mapping = qiContext.getMapping();
   FreeFrame targetFrame = mapping.makeFreeFrame();

   //Update the frame; currentFrame, targetFrame
   targetFrame.update(robotFrame, transform, System.currentTimeMillis());
}
```



Building the GoTo

Step 3

- Create your GoTo Action

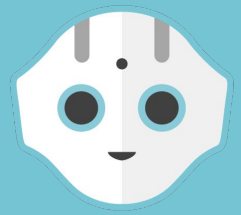
Step 4

- Run it

```
@Override
public void onRobotFocusGained(QiContext qiContext) {
        //Get the Robot Frame
    Actuation actuation = qiContext.getActuation();
    Frame robotFrame = actuation.robotFrame();

    //Move 1 Meter Forward (X Axis is forward/backward)
    Transform transform = TransformBuilder.create().fromXTranslation(1);

    Mapping mapping = qiContext.getMapping();
    FreeFrame targetFrame = mapping.makeFreeFrame();

    //Update the frame; currentFrame, targetFrame
    targetFrame.update(robotFrame, transform, System.currentTimeMillis());

    // Create the action
    goTo = GoToBuilder.with(qiContext)
            .withFrame(targetFrame.frame()) // Set the target frame.
            .build();
    // Add a consumer to the action execution.
    goTo.run()
}
```

Advanced Example - ***Free Frames***

- The previous example was relatively static and relies on hardcoded movements
- This is a sample application which lets you save locations to later return to them:
  - https://github.com/aldebaran/qisdk-tutorials
- It can be used in larger areas if Pepper needs to navigate between static points



Challenge:

Follow people around!

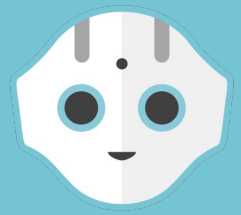
Level 1: Make Pepper follow any human she sees

- Use the fact that all “humans” from getFocusedHuman has a frame

Level 2: Make all this controlled by buttons

- A button to turn following on / off
- A button to go forwards, or to turn

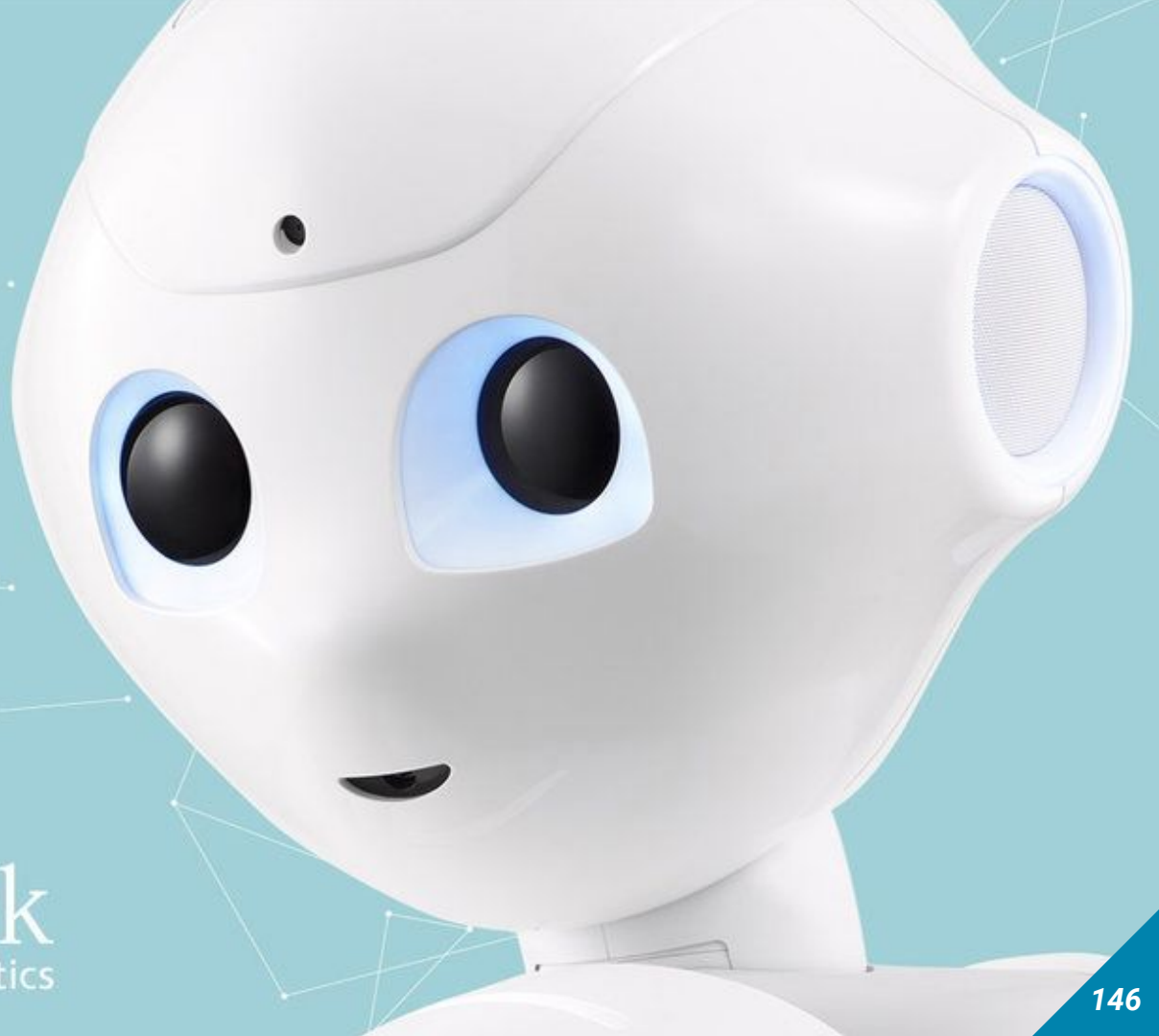
Hint: you will need to make this async



End of Day 2

Thank you!

Day 3



SoftBank Robotics

Agenda for the next 3 days

Day 1

Discover Pepper

Discover The QiSDK

Conversation

Day 2

Enter Interaction

Asynchronous Programming

Navigation

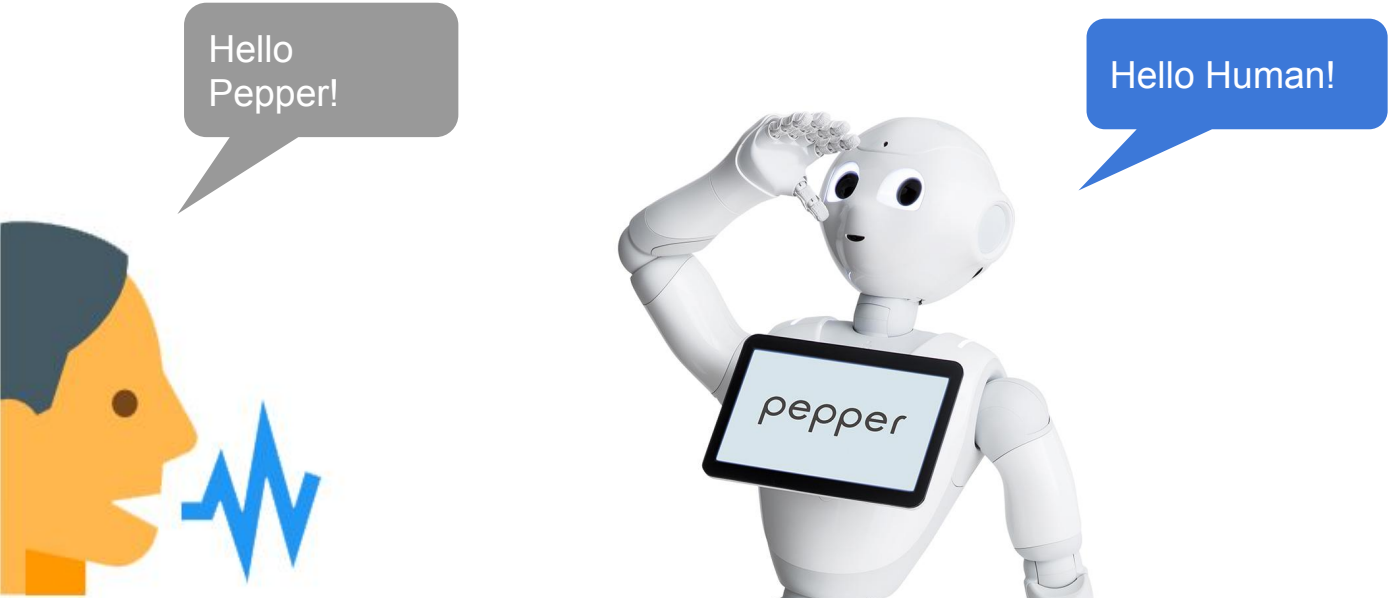
Day 3

Tips & Tricks

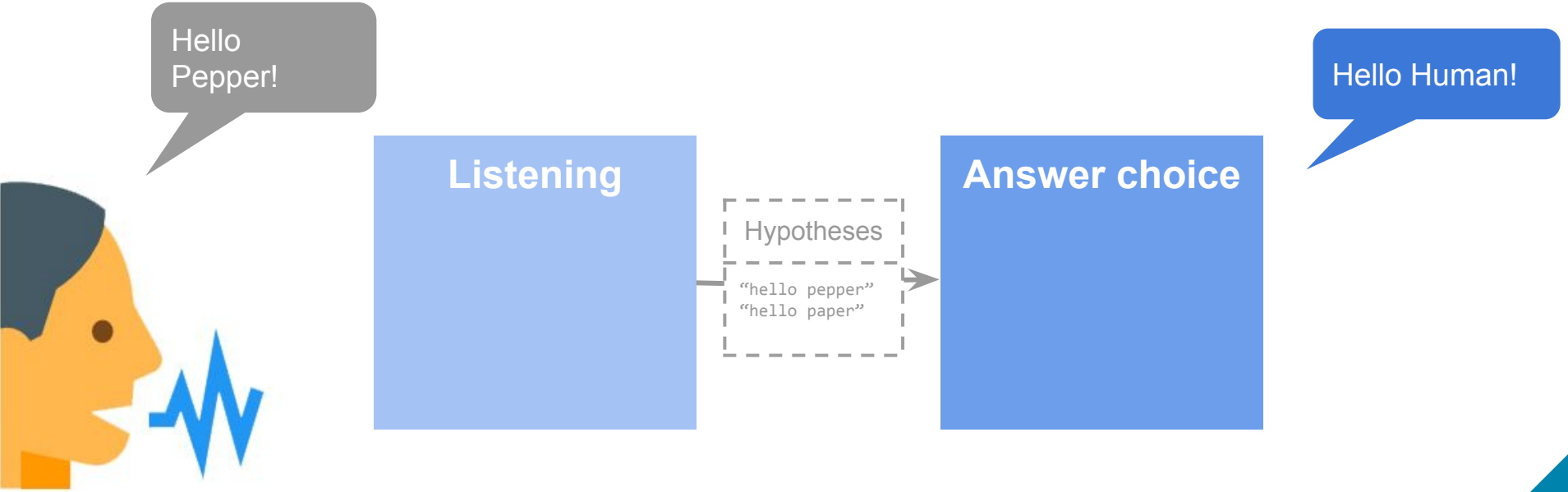
Making a full app

Theory:
How Pepper Listens

What happens when Pepper hears speech?

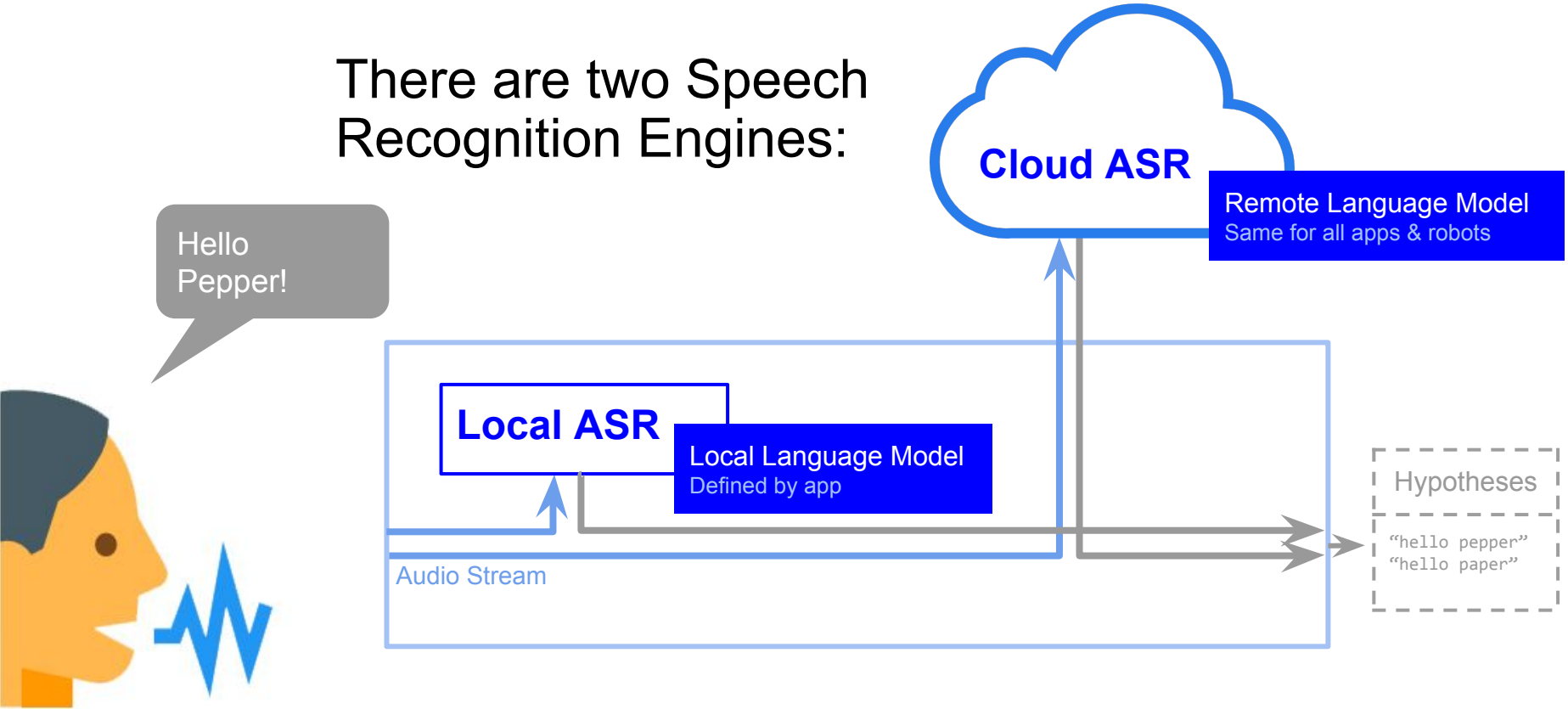


We can distinguish two parts:



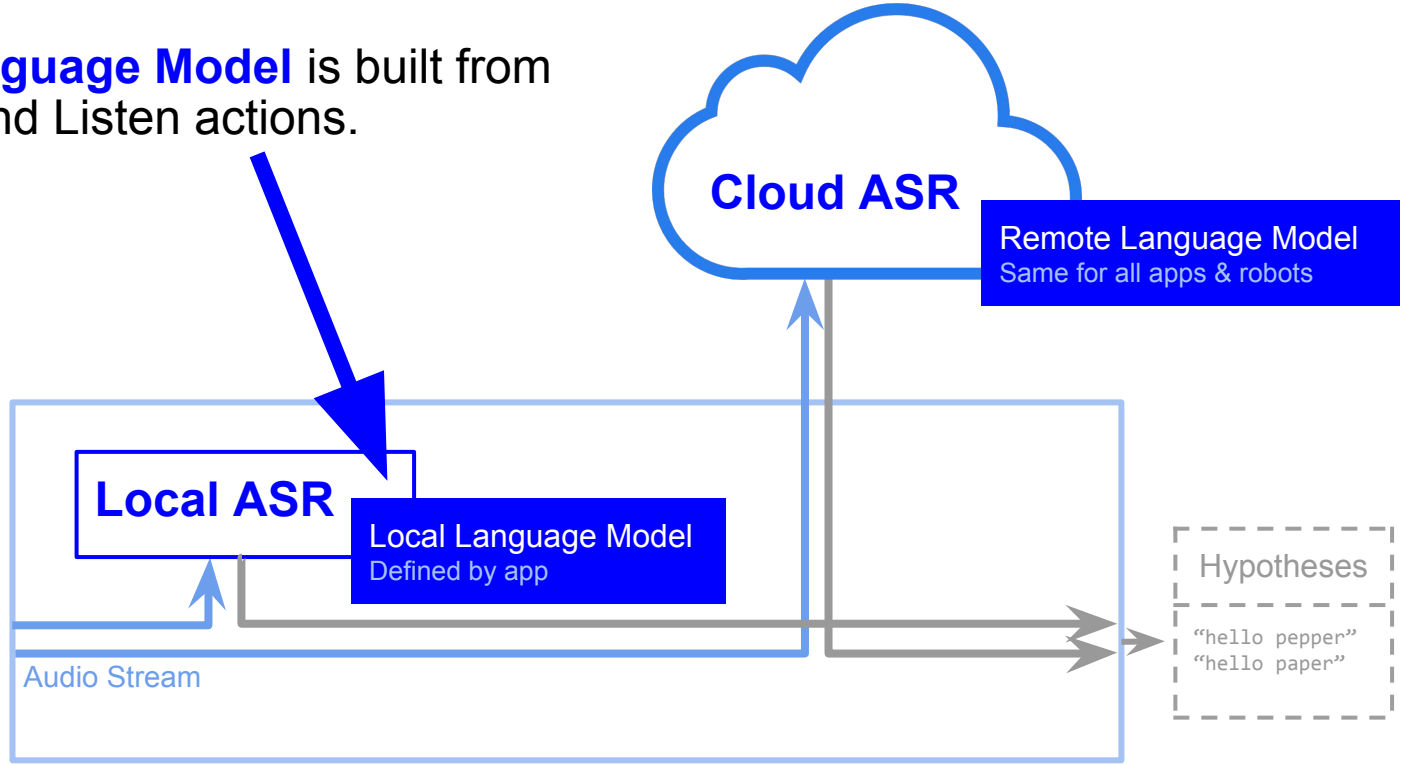
Let’s look at listening

There are two Speech Recognition Engines:



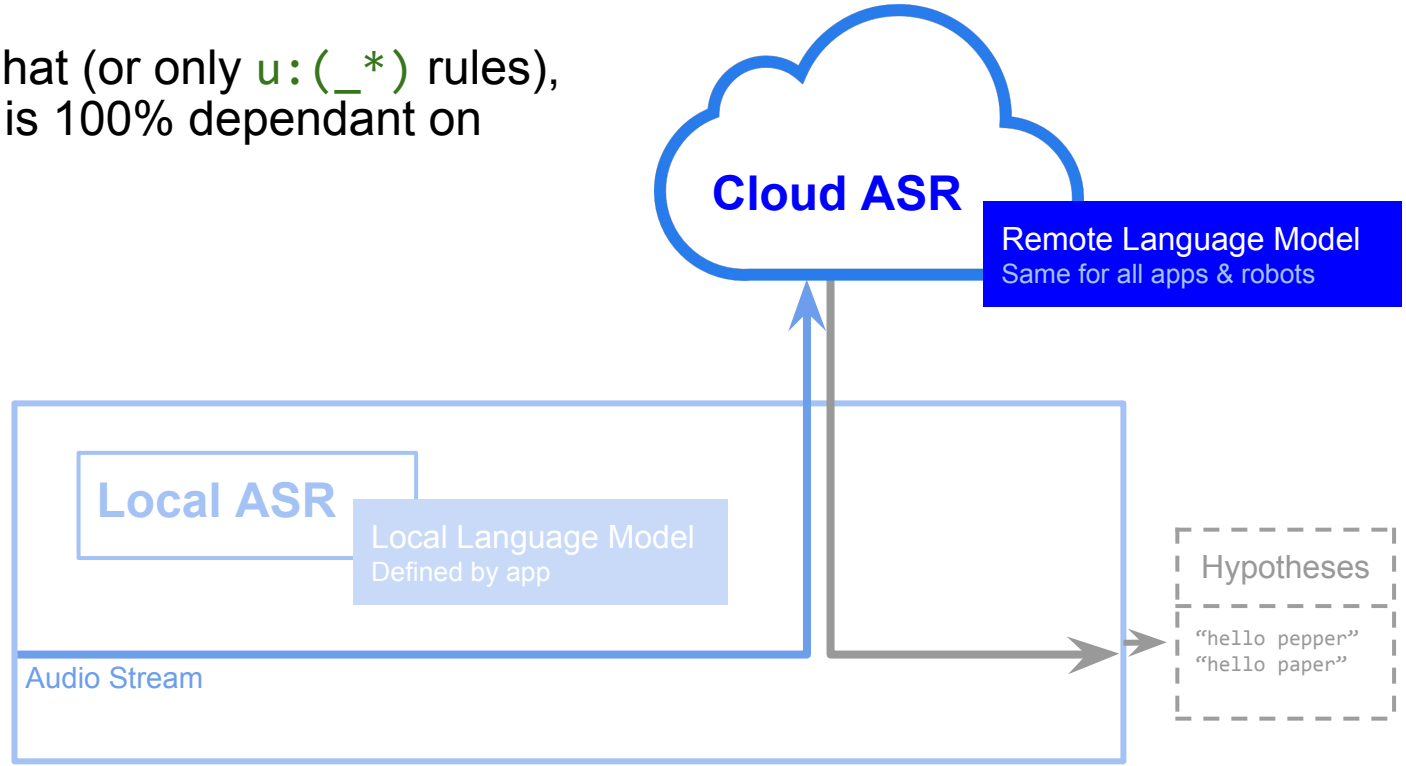
Speech Recognition: Listening

The **Local Language Model** is built from QiChat rules and Listen actions.



Speech Recognition: Listening

So with no QiChat (or only `u:(_*)` rules), the application is 100% dependant on **Cloud ASR**.



Speech Recognition: Listening

Local and Cloud ASR may produce different hypotheses:

- Local ASR can handle unusual words defined in your local language model:

User Rule in QiChat	Local ASR Hypothesis	Cloud ASR Hypothesis
`Where are the zorkons?`	Where are the zorkons ?	Where are the sores gone?

(this is often relevant for brand names, people's names, and other rare words)

- Remote ASR can fill in wildcards (Local ASR only returns <...>):

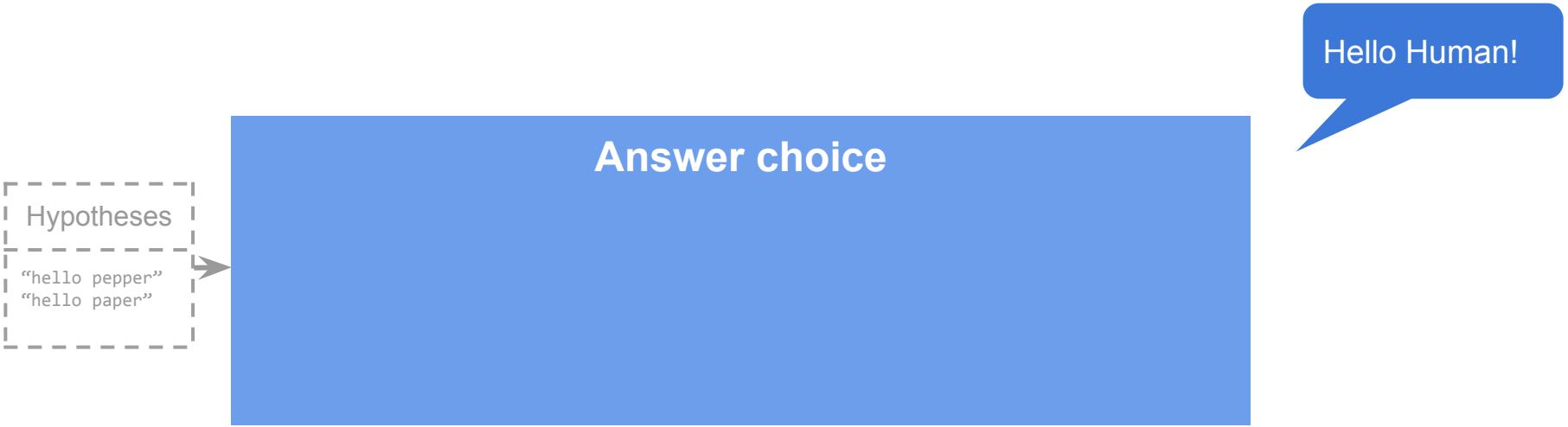
User Rule in QiChat	Local ASR Hypothesis	Cloud ASR Hypothesis
`I like robots because *`	I like robots because <...>	I like robots because they're cool

Speech Recognition: Listening

Some reasons Cloud ASR may not work:

- It's not supported for the current language (it is supported for Chinese)
- It's not enabled (In QiSDK, it's enabled at all times)
- It's not authorized
  - Is is activated robot-by-robot by SBR (Business Partners get 5000 requests/ day)
- It's not connected
  - Cloud ASR of course requires a good working internet connection
  - If connection fails or times out, Pepper will stop using Cloud ASR for 5 minutes.

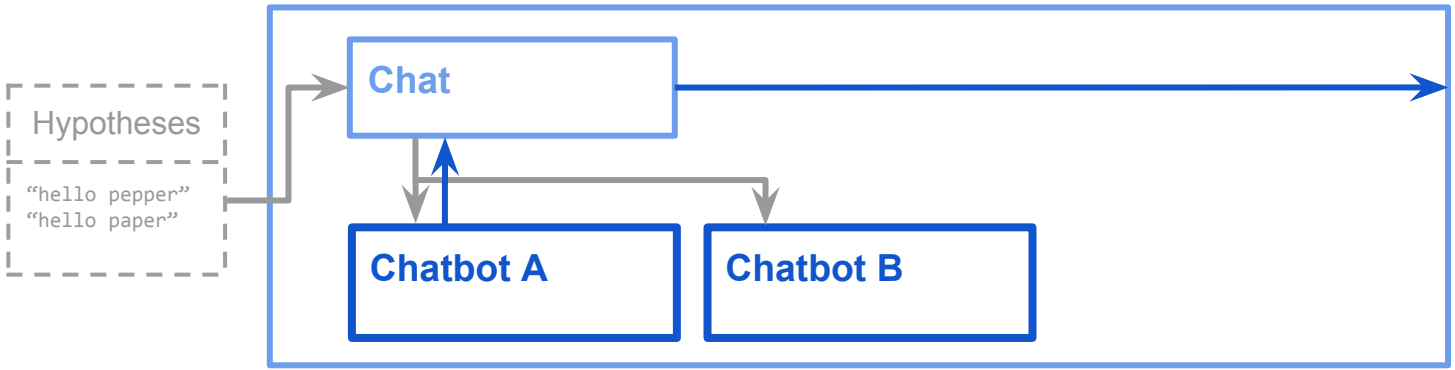
Now let’s look at answer choice



Speech Recognition: Answer Choice

There is only a single **Chat** action active, but it can contain any number of chatbots.

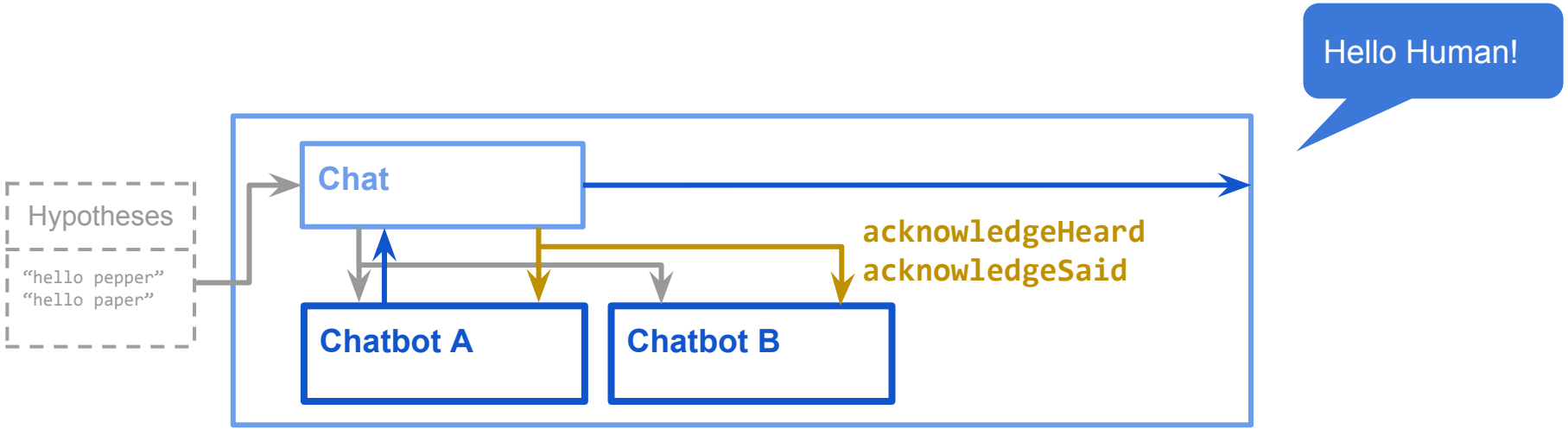
These will each evaluate the hypotheses, and decide whether to answer, with a **ChatbotReaction** (*usually* containing a Say)



Hello Human!

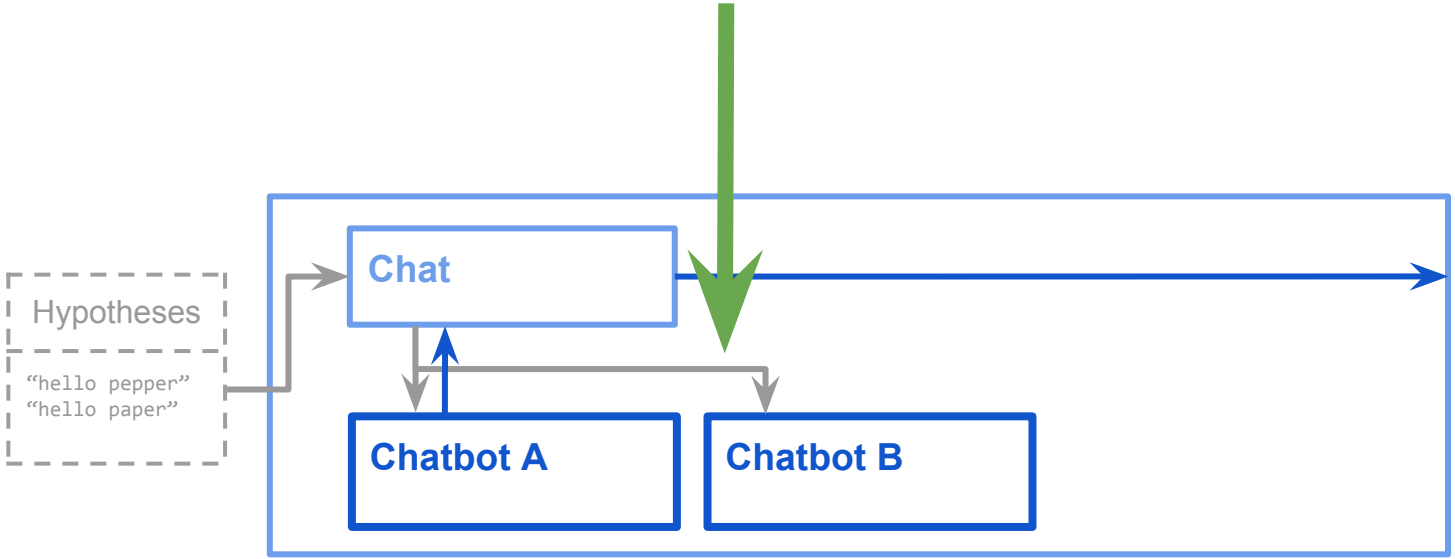
Speech Recognition: Answer Choice

Answers are selected by priority (NORMAL or FALLBACK). Once an answer is selected, all chatbots are **notified** of what it is.



Speech Recognition: Answer Choice

You can define how many hypotheses each chatbot evaluates with `chatbot.setMaxHypothesesPerUtterance` (2 by default)



Speech Recognition: Answer Choice

QiChatbot is one Chatbot implementation.

Internally, it sees if any of the hypotheses **matches** a QiChat user rule. For example:

User Rule in QiChat	Local ASR Hypothesis	Cloud ASR Hypothesis
`What did you say?`	What did you say?	What did you say?
`Where are the zorkons?`	Where are the zorkons?	Where are the sores gone?

Notice how some unusual words (like “zorkon”) will never be in the hypotheses returned by the Cloud ASR.

Speech Recognition: Answer Choice

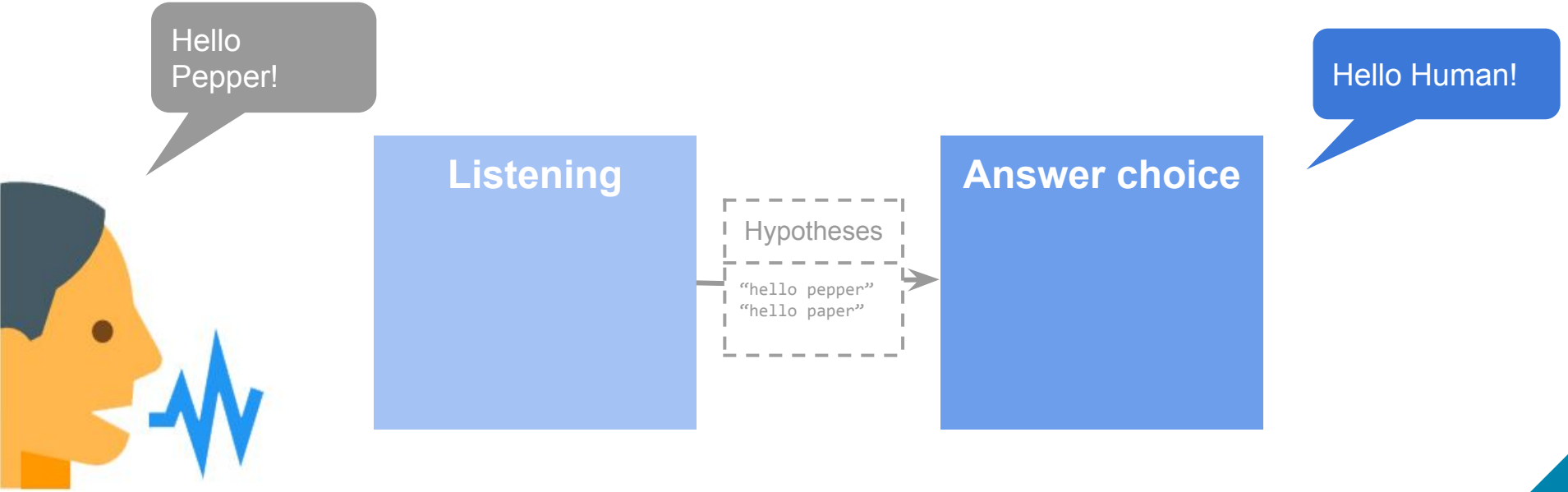
As for wildcards:

- A “bare” wildcard(*) will match anything
- A wildcard assigned to a variable (_*) will *not* match <...>

User Rule in QiChat	Local ASR Hypothesis	Cloud ASR Hypothesis
`I like robots because *`	I like robots because <...>	I like robots because they’re cool
`My name is _*`	My name is <...>	My name is Raphael
`My favourite zorkon is _*`	My favourite zorkon is <...>	My favorite soars gone is David

Notice how this means some inputs may never be matched!

What if all this doesn't seem to work ?



Troubleshooting quizz:

What do each of these speech bar states show?

The speech bar shows:
?
My hovercraft is full of eels
I like robots because <...>
I like robots because they do what I tell them to
My name is <...>

What the speech bar tells you about ASRs

The speech bar shows:	Explanation:
?	Not matched, and Cloud ASR is not active
My hovercraft is full of eels	Not matched, but Cloud ASR gave it's best guess
I like robots because <...>	Matched, and ASR is probably not active (or returned something else that didn't match anything)
I like robots because they do what I tell them to	Matched, and Cloud ASR is active
My name is <...>	Not matched (if the user rule was `u:(My name is _*)`), and Cloud ASR is either not active or didn't match anything at all (if it had returned something, that would be shown instead)

Other possible issues:

- The robot is not listening (for example, because he's talking)
- The robot is not in the right language
- The language pack is not installed on your robot
- There is a syntax error in your QiChat so it hasn't been loaded
- Your QiChat rules are not broad enough to cover what the user actually said
- The robot is not looking towards the user, and so does not pick up the sound as accurately (this is by design - we don't want the robot to react to random people)
- The user has a strong accent (understanding the voice of foreigners and children can be difficult)
- The user left a long pause in his speech, and the robot reacted to the half-sentence too early

Useful Tips

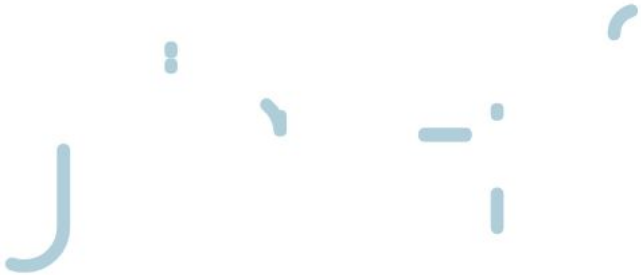
Lottie

- Easy to setup

- Tablet display is just an asset that can be changed without touching a line of code as QiAnim

- If needed, animations are easy to control within the code

- Download After Effects (Illustrator might be useful)
- Install the plugin Bodymovin (Bodymovin)
- Make your lottie
- Export the lottie
- Preview on webapp (https://lottie.cloud/app/)
- Test on a robot
- Withdraw important frames
- Integrate your lottie



“One shot”

“Controlled”

http://airbnb.io/lottie/supported-features.html

2.5

2.9

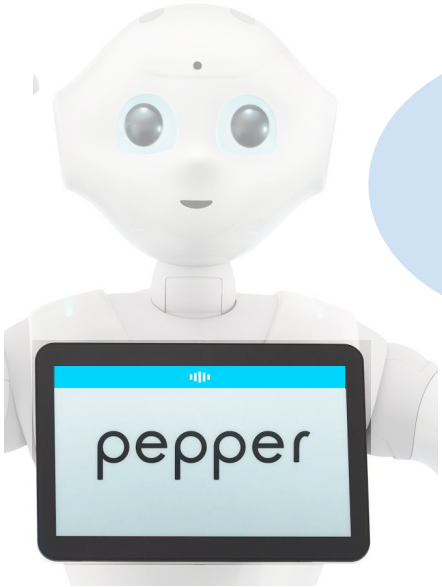
Theory

Interaction Design Tips

Remember you're making a robot interaction, not a tablet interaction !

- Use dialogue, not (just) buttons
- Make everything bigger
- Make the speech match the tablet
- Keep speech simple and short

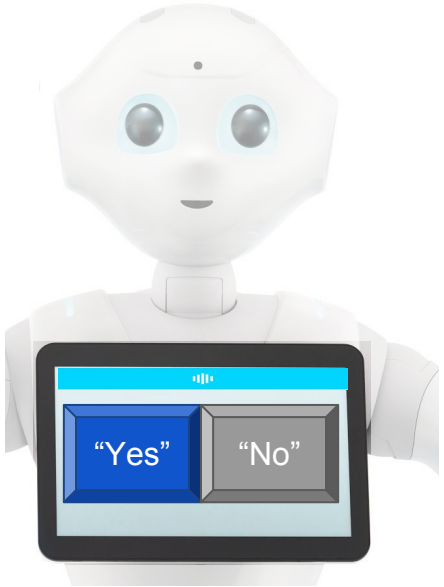
Use dialogue, not (just) buttons!



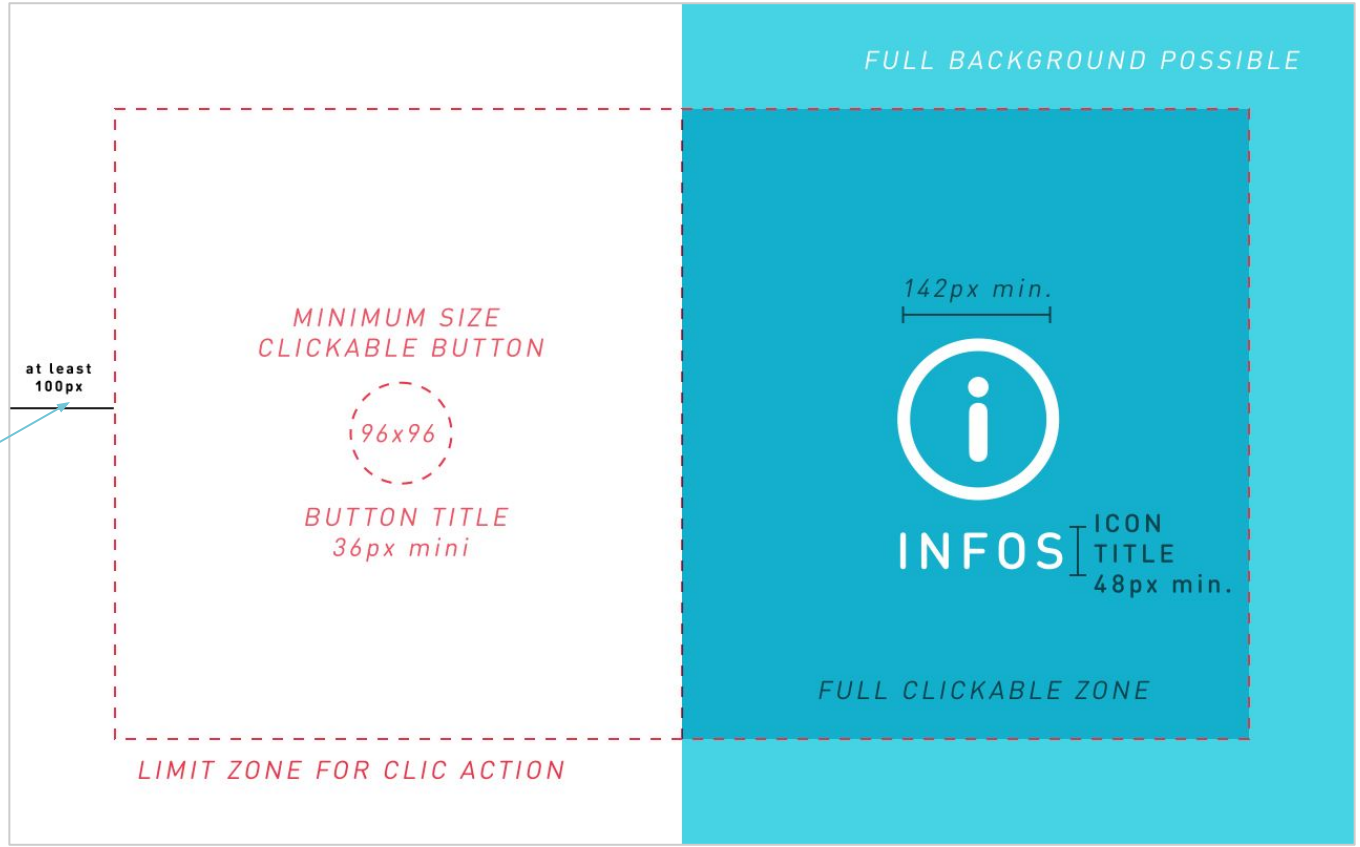
Hey Pepper, how's the weather tomorrow?

How people use a tablet vs. how people talk to Pepper

Make everything bigger



Make everything bigger



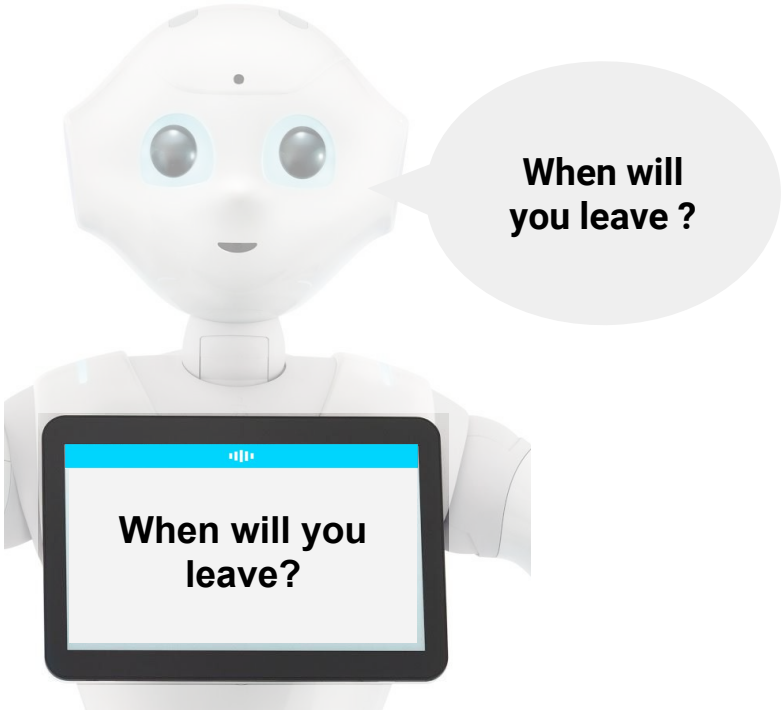
Avoid getting to close to borders

Always help users with clic actions, by making icons, buttons or clic zone bigger

8 icons menu maximum also recommended

Make the speech match the tablet

When Speaking



When Listening

Keep speech simple and short

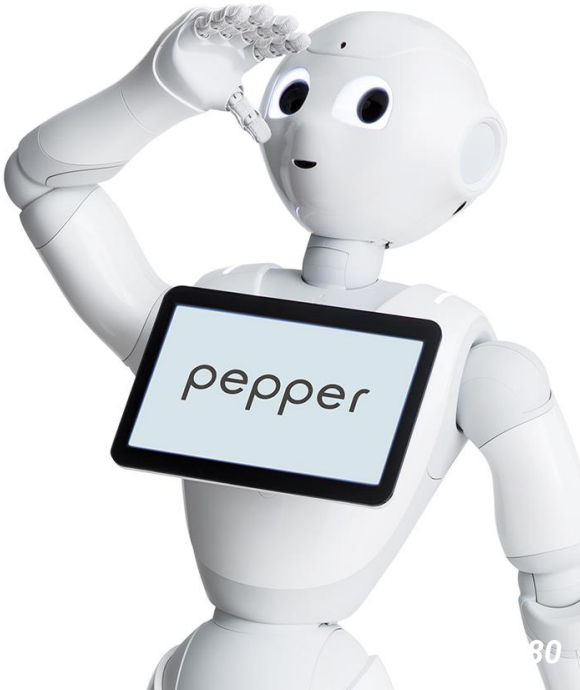
Thank you for playing; if you want to play again tell me “play again”, and if you want something else say “what’s the weather” for a weather forecast or “when is my train” for a train schedule. If you want me to repeat say “repeat”.

Do you want to play again?

For more on app interaction design:

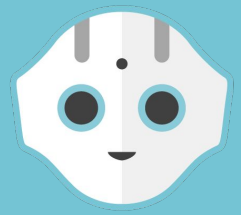
>> Pepper B2BD guidelines

On http://doc.aldebaran.com



SoftBank Robotics

pepper

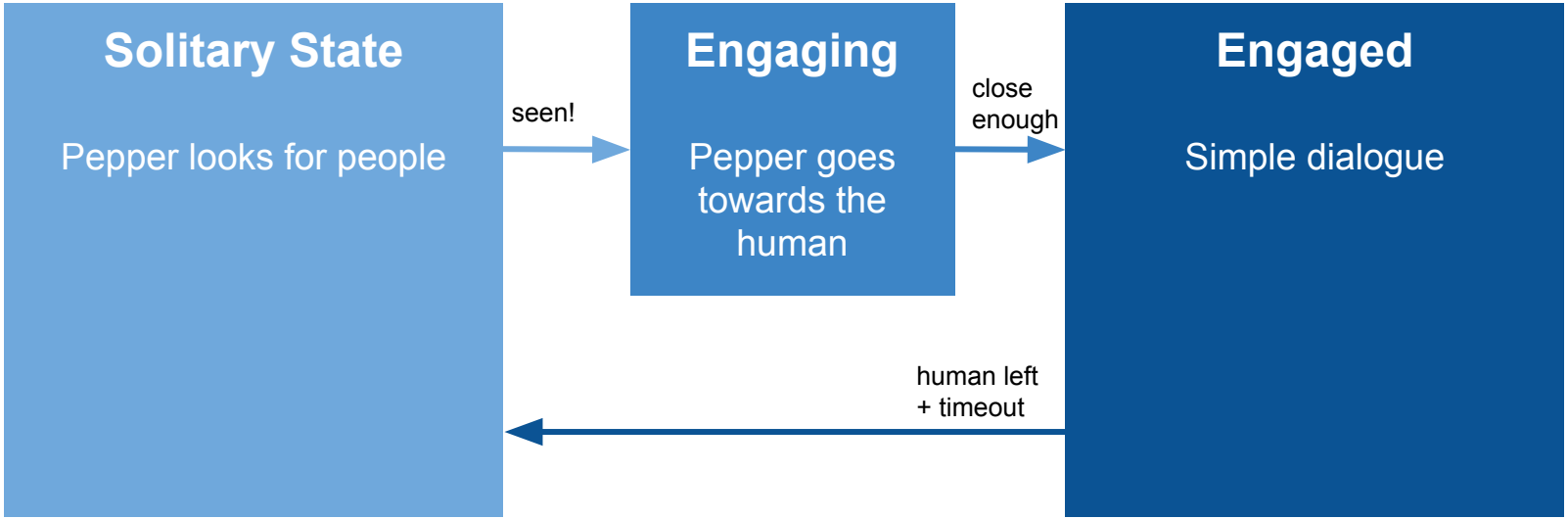


Challenge

Make a full app

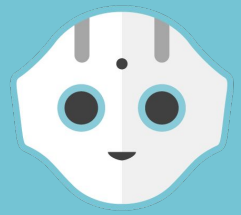
More complete application flow

Apply what you learnt on perception, motion and QiChat to build:



SoftBank Robotics

pepper



End of Day 3

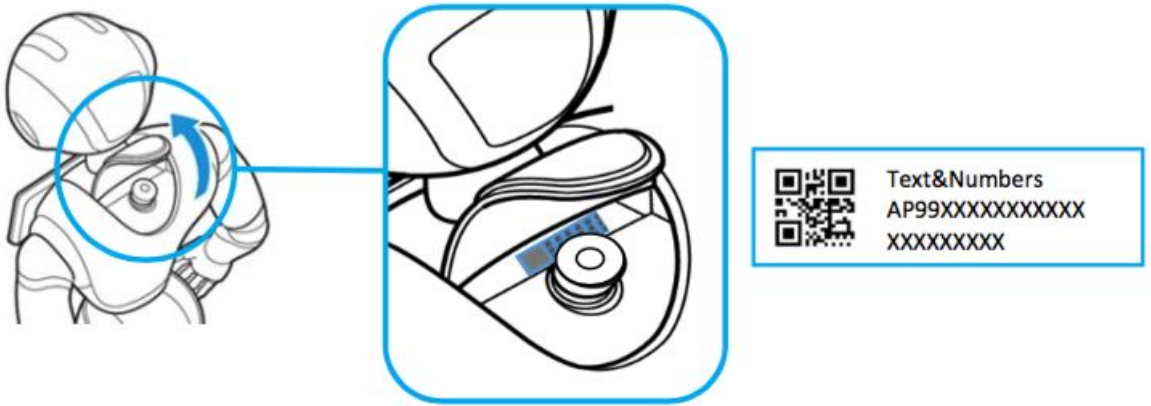
Thank you!

SoftBank Robotics

pepper

Maintenance

An Overview



- Factory reset
- Upgrade
- Customer care

- Motors or sensors misbehaving? *(e.g. wrong angle, unreachable peripheral, etc.)*
- Pepper detect and notify you of an error?

Reflash the controllers:

- Switch off
- Wait 10 seconds
- Press and hold the chest button for 8s *(shoulders turn blue; activates reflash)*
- Reflash boot will take ~15-20min

reflash /riːˈflash/ *verb* - *Boot mode that checks all the internal microcontrollers and flashes (resets) their firmwares if needed. No user information will be impacted; it's only low-level.*

SoftBank Robotics

pepper

The End

Useful Links

Android Documentation: https://developer.android.com/

QiSDK Documentation: http:///android.ald.softbankrobotics.com

Samples: https://github.com/aldebaran?q=qisdk-sample