

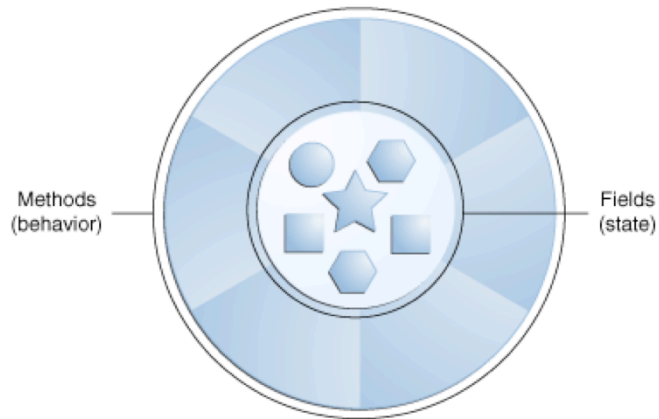
Java
JavaScript + jQuery

Filip Kis
fkis@kth.se

Object Oriented

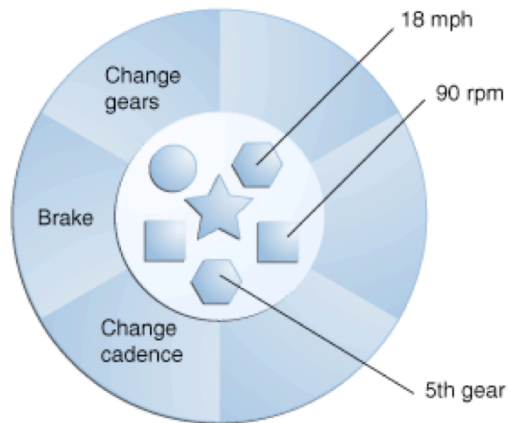
JAVA

Objects



Objects are the key of object-oriented technology. They are all around us and that is why this concept was adopted for the programming as well. Objects have two characteristics: state and behavior. In Java (and most OO languages) they are represented with fields and methods respectively.

Objects



Methods typically operate on fields to change their state. Here we can see a Bicycle as an example of an object.

```
public class Bicycle {  
  
    private int speed = 0;  
    private int gear = 1;  
  
    public void changeGear(int newValue) {  
        gear = newValue;  
    }  
    public void speedUp(int increment) {  
        speed = speed + increment;  
    }  
    public void applyBrakes(int decrement) {  
        speed = speed - decrement;  
    }  
}
```

Class

In real world we have many objects that are of the same kind. Many bicycles of the same mark and model. A class represents the definition (blueprint) of that mark and model. One bicycle is an *instance* of a *class of objects* known as bicycles. Here we see one class that defines what fields (state) and methods (behavior) a bicycle has. The fields are defined as variables of certain type (int in this case) and with some initial value (doesn't need to be defined. Here we also have some examples of methods, but we'll explain them in details later.

```
Bicycle bike1 = new Bicycle();  
Bicycle bike2 = BikeFactory.makeBike();  
  
bike1.changeGear(2);  
bike1.speedUp(30);  
  
bike2.changeGear(1);  
bike2.speedUp(12);
```

Class

To get the actual instances we need to either create them by using new keyword. Or get them from somewhere else (like in this example with BikeFactory). This is called instantiating the variable. After variables have been instantiated we can call the methods on them to change their states.

```
public class Bicycle {
```

```
    // rest of the code
```

```
    public int getSpeed() {  
        return speed;  
    }
```

```
    public void matchSpeed(Bicycle b, int gear) {  
        speed = b.getSpeed();  
        this.gear = gear;  
    }  
}
```

Methods

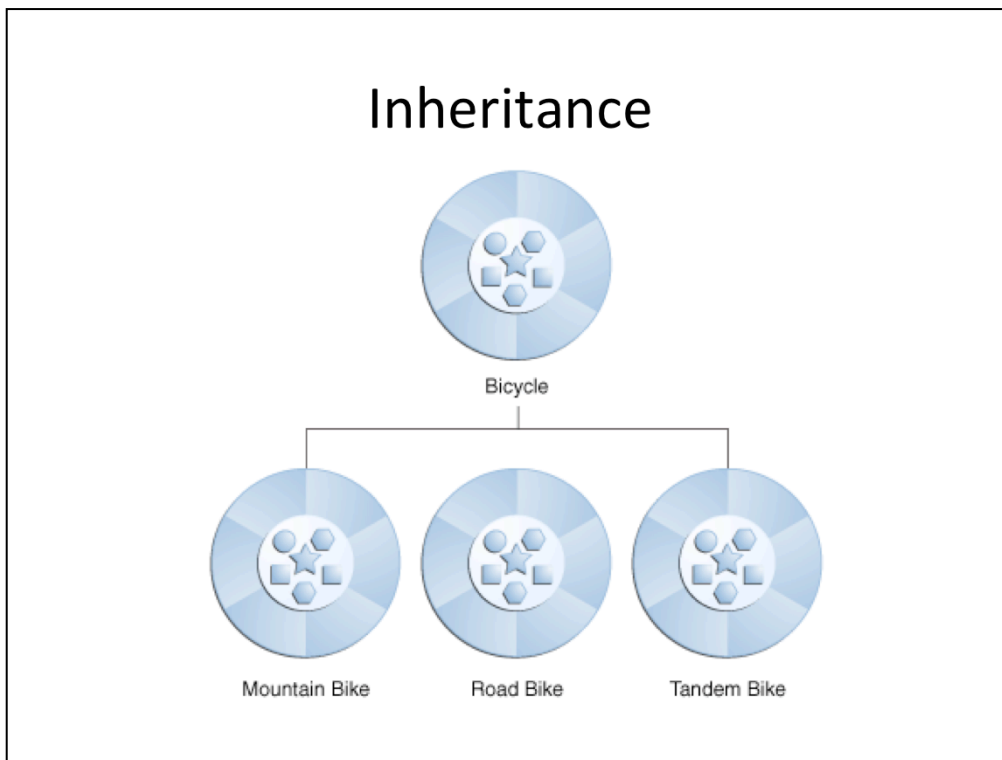
You define methods by providing a returning type (int in the first case) or void if your method doesn't return anything (second case). The methods can also have input parameters. The parameters can be either of primitive type (like int) or other objects (like Bicycle). In the body of the method we do some work and in case we have a returning type, we need to return something back. In the line "speed = b.getSpeed();" you can see how the method getSpeed returns a value and sets it to the speed variable.

Methods

```
Bicycle bike1 = new Bicycle();  
Bicycle bike2 = BikeFactory.makeBike();  
  
bike1.changeGear(2);  
bike1.speedUp(30);  
  
bike2.changeGear(1);  
bike2.speedUp(12);  
  
bike2.matchSpeed(bike1,3);
```

Here is another example of calling a method which has a Bicycle object as a parameter.

Inheritance



Different kinds of objects often have a certain amount in common with each other. Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes. In this example, Bicycle now becomes the *superclass* of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*.

```
public class Bicycle {  
    //the rest of the class code  
}  
  
public class RoadBicycle extends Bicycle{  
  
    private boolean light;  
  
    public void setLights(boolean lightsOn) {  
        light = lightsOn;  
    }  
  
}
```

Inheritance

Here we see a RoadBicycle that is a subclass of Bicycle. It adds another field (light) and another method (setLights).

Inheritance

```
RoadBicycle road1 = new RoadBicycle();  
  
road1.changeGear(2);  
road1.speedUp(30);  
road1.setLights(true);
```

Here we can see how we create a new object of the new Bicycle subclass. As you can see we can access all the methods from its base class (changeGear and speedUp) as well as its specific method (setLights).

Inheritance

```
public class BikeFactory {  
  
    public static void fixBicycle(Bicycle b) {  
        //code for fixing the bicylce  
    }  
    //the rest of the class code  
}  
  
BikeFactory.fixBicycle(bike1);  
BikeFactory.fixBicycle(road1);
```

Another important thing to understand about inheritance is that a RoadBicycle is still a Bicycle. In other words when you have a method that works with some class that you are inheriting (as in above example fixBicycle expects Bicycle parameter), you can pass your inherited class (in our example we can pass RoadBicycle to the methods expecting Bicycle).

Constructor

```
public class RoadBicycle extends Bicycle{  
  
    private boolean light;  
  
    public RoadBicycle(int initialSpeed) {  
        speedUp(initialSpeed);  
        light = false;  
    }  
  
}
```

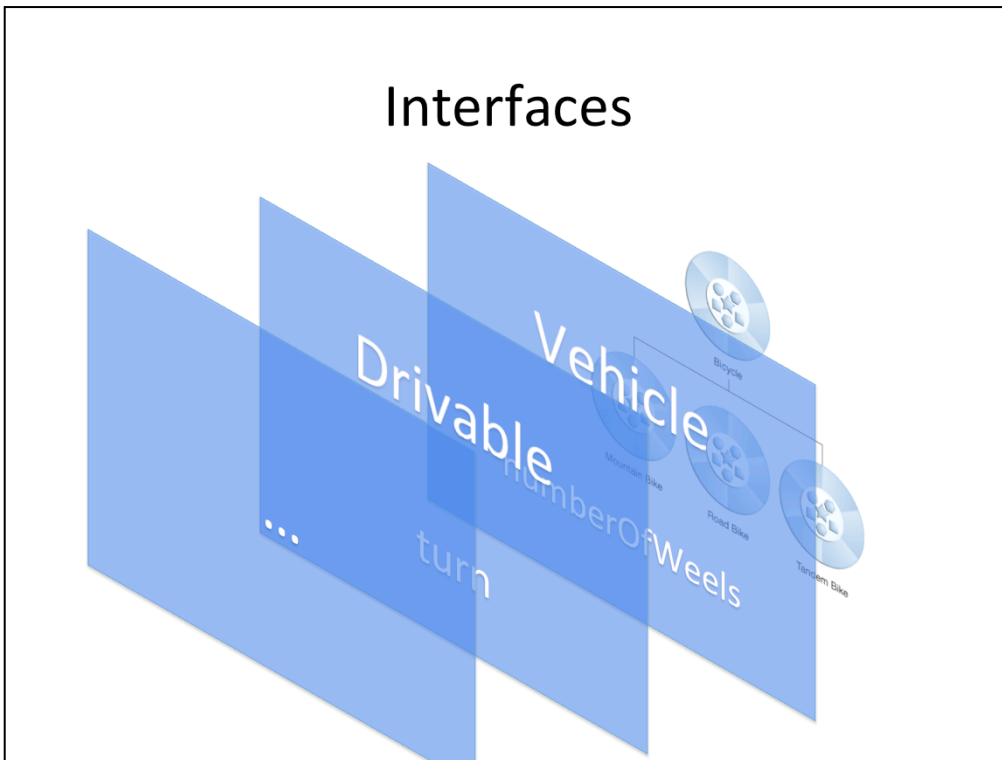
Constructor is a special kind of method (as you can see it doesn't have a return type and it must be of the same name as the class). It is used to create a instance of that class and set some specific values. This is useful when you need some input on how the object should be created or you want to provide an easy way of setting default values (instead of calling all the methods one by one each time the user creates the object).

Constructor

```
RoadBicycle road1 = new RoadBicycle(20);  
  
road1.changeGear(2);  
road1.speedUp(30);  
road1.setLights(true);
```

Here you can see how you use the constructor. You can't anymore use the empty constructor (see slide 11), but you have to pass the value of initial speed when you're constructing RoadBicycle.

Interfaces



As you've already learned, objects define their interaction with the outside world through the methods that they expose. Sometimes you want to define what behavior should exist, but you don't know yet how each specific class will implement that behavior. That's when interfaces come into play. They define the expected methods of a class that is implementing them, but not the behavior (the method bodies are empty). One class can implement any number of interfaces.

```
public interface Vehicle {
    public int numberOfWheels();
}

public class Bicycle implements Vehicle {
    public int numberOfWheels() {
        return 2;
    }
    // rest of the code
}

public class Car implements Vehicle {
    public int numberOfWheels() {
        return 4;
    }
    // rest of the code
}
```

Interface

Here you can see an example of Vehicle interface that expects the class using them to implement a method numberOfWheels. Each class that wants to be considered a Vehicle (and be treated as one) will need to implement that interface. When you implement an interface you must provide the actual code for all the methods that the interface defines. Here we see Bicycle and Car class implementing Vehicle interface and its method numberOfWheels.

Interface

```
public void tellMeNumberOfWheels(Vehicle v) {  
    System.println(v.numberOfWheels());  
}  
  
tellMeNumberOfWheels(bike1);  
tellMeNumberOfWheels(car1);
```

You can then have method that expects a class that implements a certain interface. You can then call the interface method on the object passed and be sure that this method exists. If you didn't implement the interface, but you still had the method with the same name, this wouldn't be enough for the system. The interface is a form of contract that guarantees that the method will exist and that it will work as expected.

Package

```
package se.kth.csc.iprog;
```

```
import java.util.*;
```

```
import java.lang.Math;
```

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. It helps you organize your code and put in some logical order.

When you want to use classes from different packages you import the whole package (as example `java.util.*`) or just a specific class that you need (`java.lang.Math`).

Data Types

```
byte a = 12; //-128 to 128
short b = 42; //-32,768 to 32,767
int c = 200; // a loot
long d = 767; // even more
float e = 3.42; // floating point
double f = 7.64; // double percision
boolean g = true; // true/false
char h = 'C'; // unicode character

String i = "Some string";
```

Here is a list of basic data types and how you represent their constants. Although String is not a basic type (but actually a class) it is very well supported internally in java and acts almost as basic type.

Control and Loop

```
if ( a == 1 ) {  
    // do something  
} else {  
    // something else  
}
```

```
for (int i = 0; i < 40; i++) {  
    // do something  
}
```

while, do-while, switch

And of course, like in many other programming languages, you have loops and control flow syntax that you can use in your code. Here are some examples and you can lookup more in the resources provided on the next slide.

Resources

<http://docs.oracle.com/javase/tutorial/>

```
public interface Student {  
    public String question();  
}
```

Questions? Check the resources and if you have questions feel free to contact me.

Mocha, LiveScript, ECMAScript

JAVASCRIPT

How it came to be

Introduced in **1995**

Official name **ECMAScript**

No real connection to Java

It was developed for Netscape in 2005 and first called Mocha, then LiveScript and finally JavaScript. The name came from collaboration of Netscape and Sun on incorporating Java in web. When it became standardized it's official name became ECMAScript.

It's a very powerful language that's often misunderstood because of some of it's trades. It's not a script (but a scripting language). It's not in any way connected with Java apart from Java being an inspiration when it was developed.

What is it

Very powerful **language**

Weakly typed

Classless, **prototype** based OO

It is a very powerful language with many functionalities found in the big compiled languages, and some not found even there. It is weakly typed (meaning objects can basically be of any type and change type without any problem). This accounts for part of the confusion associated with using it. It is also a classless, prototype based OO language meaning instead of having classes of objects it uses prototypes of objects that can then be cloned.

What is it used for

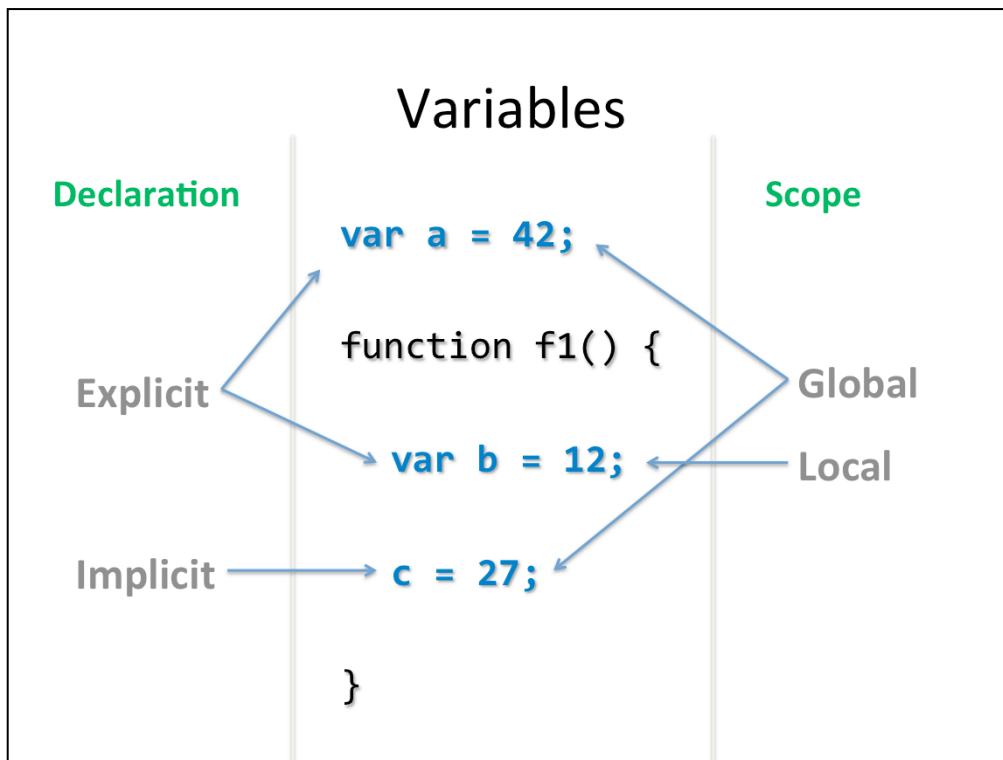
Client-side dynamic content

a.k.a.

interaction

Not only for HTML

Its main usage is for client-side dynamic content or in other words interaction. Reacting to user input, animation, data manipulation are just some of the examples of interaction that can be achieved with JavaScript. It can also be used outside of browser content as a scripting language for instance the Node.js that is lately gaining on popularity, but that's outside of scope of this course.



The variables should always be defined explicitly (using the var keyword). If you declare a variable implicitly, the variable has a global scope and that is one of the most common sources of bugs in JavaScript codes.

Data types

```
var myInt = 7;  
var myBool = true;  
var myArray = new Array();  
var myString = "ABC";
```

```
var myFunction =
```

```
var myObject =
```

A bit later

Here you have some examples of constants of different type and how you define them. The functions and objects will be mentioned a bit later.

Control and Looping

```
var a = 0;
for (var k = 1; k < 5; k ++ ) {
    a = a + k;
}

if ( a == 42 ) {
    b = "The ultimate answer";
} else {
    b = "Something else";
}
```

Loops work in the same way as in Java or C. There is also switch and while loop.

= VS. ==

```
var a = 0;
for (var k = 1; k < 5; k ++) {
  a = a + k;
}

if ( a = 42 ) {
  b = "The ultimate answer";
} else {
  b = "Something else";
}
```

One important difference from Java to notice is the meaning of = in boolean expressions. While this code would not compile in Java, in JavaScript (as in C) it is valid. It means "a becomes 42" and since everything that's not 0, false or null is true, this will always be true. No matter of what was the value of a before. Another common mistake in JavaScript, so be careful.

Functions

```
function add(a,b) {  
    return a + b;  
}  
  
c = add(3,7);
```

Since JavaScript is weakly typed, there is no return type of the function nor there is type definition of the function parameters. They will be whatever they are assigned. And the function will return whatever it returns.

Objects

```
var person = new Object();  
  
person.name = "John";  
person.surname = "Smith";  
person.age = 50;
```

Defining objects is rather simple. You just define the new Object variable and add fields to it by assigning some value to them. The fields are created automatically. You can add them at any time to any object.

Object constructor

```
function Person(name, surname, age) {  
  this.name = name;  
  this.surname = surname;  
  this.age = age;  
}  
  
var jack = new Person("Jack", "Sparrow", 112);
```

To make the code more clear and Object Oriented you can create an constructor function for your object. This keywords represents the new object.

Object methods

```
var fullName = function () {  
    return this.name + " " + this.surname;  
}  
  
function Person(name, surname, age) {  
    this.name = name;  
    this.surname = surname;  
    this.age = age;  
    this.getFullName = fullName;  
}  
  
var jack = new Person("Jack", "Sparrow", 112);  
  
jack.getFullName();
```

The methods of objects are just normal functions that are assigned to the objects. Notice how when you want to assign the function you use it without (). This practically means that the function itself is actually an object.

Object methods (alt.)

```
function Person(name, surname, age) {
  this.name = name;
  this.surname = surname;
  this.age = age;
  this.getFullName = function() {
    return this.name + " " + this.surname;
  }
}

var jack = new Person("Jack", "Sparrow", 112);

jack.getFullName();
```

Alternatively, you can define the method directly in the constructor.

Arrays

```
var a = new Array(); // empty array
var b = new Array("dog", 3, 8.4);
var c = new Array(10); // array of size 10
var d = [2, 5, 'a', 'b'];

c[15] = "hello";
d.push("hello");

var e = new Array();
e["key"] = "value";
e["numKey"] = 123;
```

There are many ways you can create arrays. Arrays store any object (they don't have to be of the same type – again because it's weakly typed language). You can assign the value of a member of an array by accessing it through its index (counting starts from 0). The arrays can also be associative meaning that instead of the index you can use a string to define its "position". This is like Maps in Java.

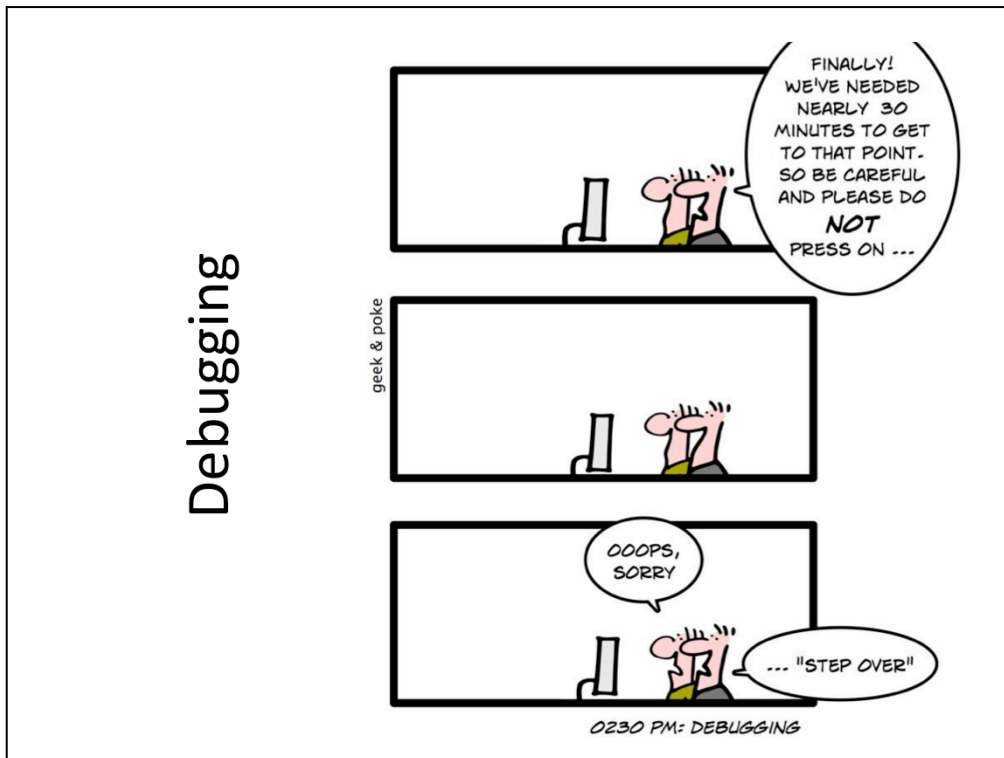
In HTML

```
<script type="text/javascript">  
  //here comes the code  
</script>
```

```
<script src="path_to_external_file.js"  
type="text/javascript"></script>
```

You use the javascript either by directly writing it in your HTML file (first example) or by including it from an external file (second example).

Debugging



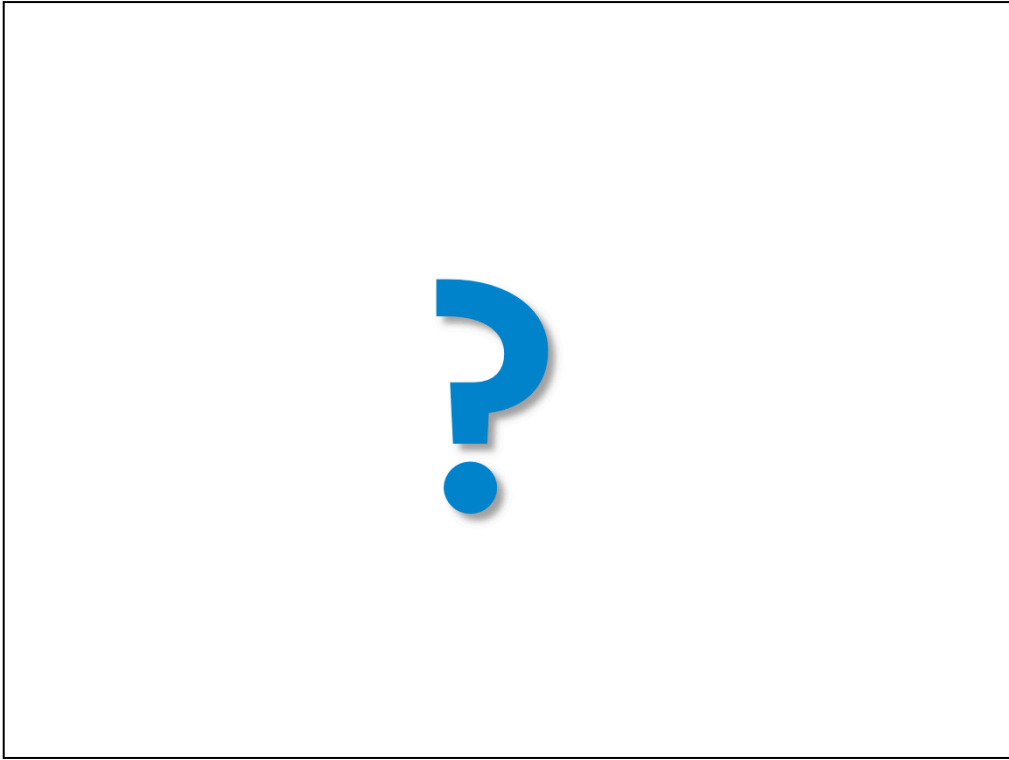
In Chrome browser (and Firefox and Internet Explorer, but they work differently) you have a good support for debugging and testing your javascript code. You activate the developer tools by Ctrl + Shift + I. There you can set breakpoints to your code, see variables and expressions and even have a shell where you can write javascript statements and test their execution.

Resources

<http://www.w3schools.com/js/default.asp>

For debugging check out:

<http://code.google.com/chrome/devtools/docs/overview.html>



Remember

Always use `var` when declaring variables

Use `==` when comparing objects

Don't forget this part, it's source of most bugs in javascript code.

Improve your quality of life

JQUERY

What is it

Lightweight JavaScript **library**

Cross-platform (browser)

Makes HTML **interaction easy**

It has been developed in 2006 by a developer from Mozilla. Its main purpose is to simplify the complicated interaction with HTML in JavaScript. Although implementations of JavaScript engine vary from browser to browser (slightly), jQuery works around these differences and for this reason is great for making sure your code works on all browsers.

What is it used for

Manipulating HTML

Animation

AJAX

...

It is mainly used to manipulate HTML (add elements, change their properties, etc.). It also has some support for animation of elements (like sliding, fade-in, fade-out). It eases doing asynchronous post-backs (AJAX) and with its very active community and big list of extensions and plugins can do much, much more.

A simple example

```
<div id="myDiv">My Div</div>
<a href="#" onclick="toggleVisibility('myDiv');">Click
here to toggle visibility of My Div</a>

function toggleVisibility(id) {
  var e = document.getElementById(id);
  if(e.style.display == 'block' || e.style.display == "")
    e.style.display = 'none';
  else
    e.style.display = 'block';
}
```

Here you can see an example of how you would do a basic thing like toggling (showing and hiding) a div element of a simple HTML page with basic JavaScript.

A simple example

```
<div id="myDiv">My Div</div>
<a href="#" onclick="toggleVisibility('myDiv');">Click
here to toggle visibility of My Div</a>

function toggleVisibility(id) {
    $('#myDiv').toggle();
}
```

Here is the same thing with jQuery. Notice the difference?

The `$()` function

```
var el = $("<div/>");
```

```
$(window).width();
```

```
$("div").hide();
```

The `$()` function represents the core of jQuery. It can be used to create new elements or manipulate existing elements.

The Philosophy

GET ACT

```
$(“div”).hide();
```

```
$(“div”).show()  
    .addClass(“main”)  
    .html(“Hello jQuery”);
```

The philosophy of jQuery is to get the object (html element) and act on it. Also, any action done it returns again an jQuery object (often the same one, but depends on this action which was performed) on which you can do other actions. This is called chainability.

Selectors

```
<div id="myDiv" class="box1">My Div</div>
```

`$("div")`

By Tag

`$("#myDiv")`

By ID

`$(".box1")`

By Class

`$("div.box1")`

You can get elements by using different selectors. If only one object satisfying the criteria is found, it is returned. Otherwise a jQuery array (not a typical array because it has some added functionality – look at the jQuery documentation for more info) of objects is returned. Here are shown only the basic selectors, while there are many, many more by which you can select, for example, only elements having some special attribute value.

HTML manipulation

```
$(“div”).html(“I’ve changed the text”);
```

```
$(“div”).css(“background-color”, “red”);
```

```
$(“ul”).prepend(“<li>Hello</li>”);
```

Here you can see some simple examples of how you can change the content of a div tag (1), change it's background (2) even add elements to a list (3).

Events

```
$(document).ready(function(){  
    // your code  
});
```

“ready” event is the main event you should call on the document before doing any other manipulation. It fires when the whole document is ready (meaning everything has been loaded). You should put all your code inside this method so to make sure that all the elements that you might be using in your code have actually been created already.

Events

```
$("#myButton").click(function(){  
    alert("Button clicked!");  
});  
$("#myButton").click();
```

You can simply add events by calling the method of the event you want to create behavior for and giving it the function that should represent the expected behavior.

You can also trigger the behavior by just calling the method (without arguments).

Events

```
var buttonClick = function() {  
    alert("Button clicked!");  
}  
  
$("#myButton").click(buttonClick);
```

This is just another way of creating the event handler (by using the already learnt possibility of javascript to create variables that represent functions).

Effects

```
$(“div”).toggle(100);
```

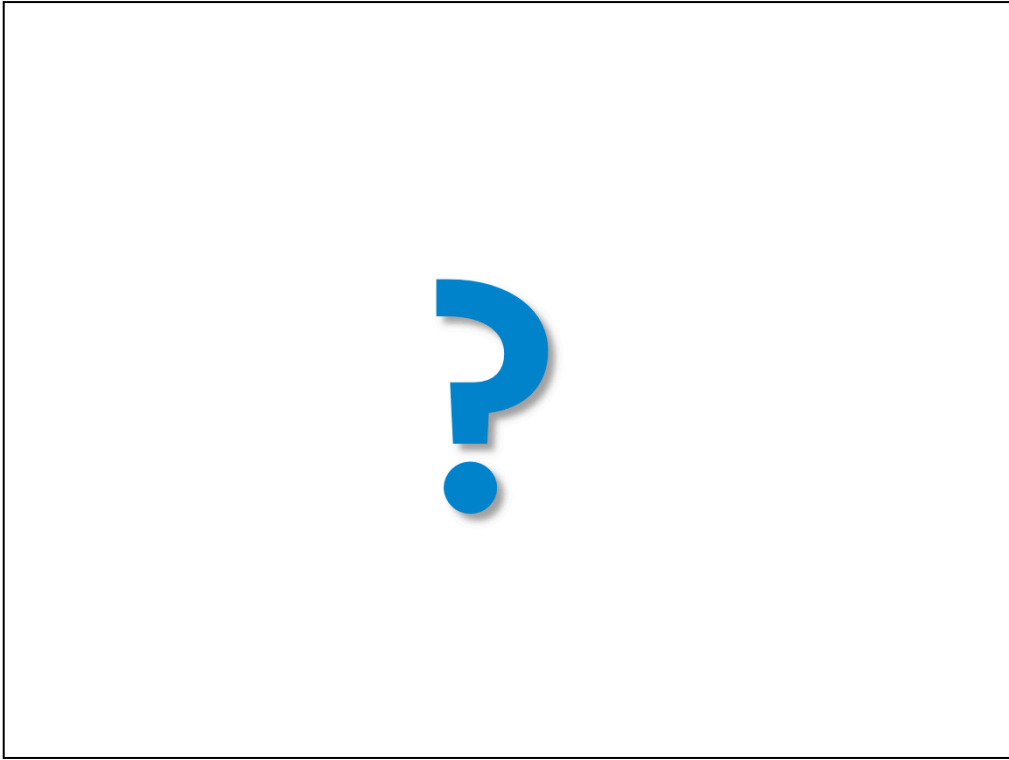
```
$(“div”).slideToggle(1000);
```

Resources

jQuery web-site: <http://jquery.com>

jQuery API: <http://api.jquery.com>

jQuery tutorial: <http://www.w3schools.com/jquery/default.asp>



Java for mobile – kind of

ANDROID

How it came to be

Founded in **2003**

Bought by **Google** in 2005

Launched the first phone in 2007

It was founded by Andy Rubin with the goal to create “smarter mobile devices that are more aware of its owner's location and preferences”, but not much was known.

In 2005 Google bought the company and continued working on the project secretly. In 2007 Google, together with phone manufacturers, wireless carriers and chipset makers announced an Open-Handset Alliance with the goal of developing open standards for mobile devices. At the same time they announced the first Android based device.

What is it

Open Source OS based on **linux**

Application written in **Java**

Most used smartphone platform

Android is an Open-Source Operating System that is based on linux (the core of the OS). It has a virtual machine that runs the byte-code application, or in other words programs written in Java.

Currently it is the most used smartphone platform since it overtook Symbian in 2010.

What do we need

Android SDK – needed and useful tools

Eclipse IDE – development environment

API – set of libraries for specific version

Android SDK provides a set of useful tools for development:

Emulator – allows for running the developed application on the computer instead of actual physical device. Useful for faster debugging and for simulating different cases (screen sizes, android versions, etc.). You can configure Android Virtual Device (AVD) through user-friendly interface. There you can even simulate certain conditions (battery level, storage space, gps coordinates, etc.).

Logcat – collects logs from the device

Debugger

And more...


There is plugin for Eclipse IDE (the most popular open source IDE) which brings the Android SDK tools and graphical editors for screens.

APIs are libraries for different Android versions. As Android developed new features were added. For this reason, when you develop an application you need to specify what is the minimum version you support and should not use any of the features that are present only in the versions higher than the one you support.

What's an Android Project?

AndroidManifest.xml – app description

 **src/** – your classes (remember java) go here

 **res/** – app resource, like **layout**
descriptions, string constants, etc.

AndroidManifest.xml - The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll learn about various declarations in this file as you read more training classes. One of the most important elements your manifest should include is the `<uses-sdk>` element. This declares your app's compatibility with different Android versions using the `android:minSdkVersion` and `android:targetSdkVersion` attributes

src - Directory for your app's main source files. By default, it includes an Activity class that runs when your app is launched using the app icon.

res - Contains several sub-directories for app resources. Here are just a few:

drawable-hdpi/

Directory for drawable objects (such as bitmaps) that are designed for high-density (hdpi) screens. Other drawable directories contain assets designed for other screen densities.

layout/

Directory for files that define your app's user interface.

values/

Directory for other various XML files that contain a collection of resources, such as string and color definitions.

main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send" />
</LinearLayout>
```

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects. View objects are usually UI widgets such as buttons or text fields and ViewGroup objects are invisible view containers that define how the child views are laid out, such as in a grid or a vertical list.

android:id

This provides a unique identifier for the view, which you can use to reference the object from your app code, such as to read and manipulate the object (you'll see this in the next lesson).

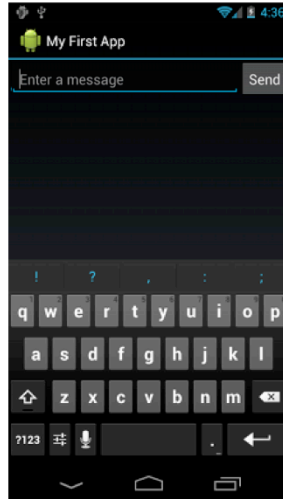
The at sign (@) is required when you're referring to any resource object from XML. It is followed by the resource type (id in this case), a slash, then the resource name (edit_message).

The plus sign (+) before the resource type is needed only when you're defining a resource ID for the first time. When you compile the app, the SDK tools use the ID name to create a new resource ID in your project's gen/R.java file that refers to the EditText element. Once the resource ID is declared once this way, other references to the ID do not need the plus sign. Using the plus sign is necessary only when specifying a new resource ID and not needed for concrete resources such as strings or layouts. See the sidebar for more information about resource objects.

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
  <string name="app_name">My First App</string>  
  <string name="edit_message">Enter a message</string>  
  <string name="button_send">Send</string>  
</resources>
```

This is how it looks



MainActivity.java

```
package com.example.myfirstapp;

import android.app.Activity;
import android.os.Bundle;

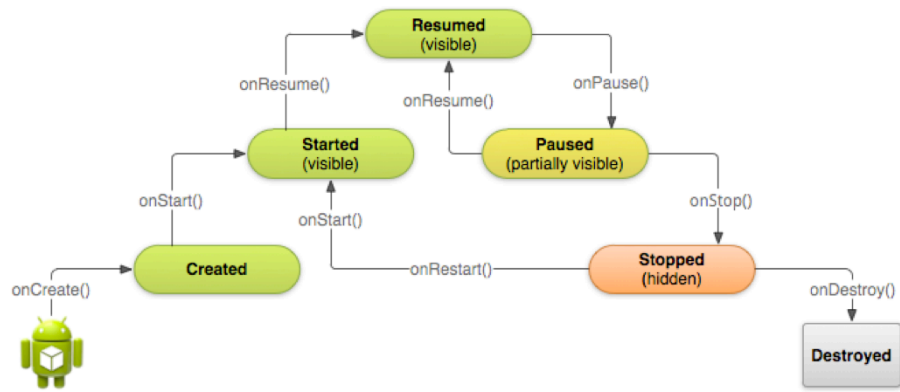
public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Each activity represent one fullscreen window. Some activities can have a return value (like dialog boxes).

As a user navigates through, out of, and back to your app, the Activity instances in your app transition between different states in their lifecycle. For instance, when your activity starts for the first time, it comes to the foreground of the system and receives user focus. During this process, the Android system calls a series of lifecycle methods on the activity in which you set up the user interface and other components. If the user performs an action that starts another activity or switches to another app, the system calls another set of lifecycle methods on your activity as it moves into the background (where the activity is no longer visible, but the instance and its state remains intact).

Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot.

Activity lifecycle



main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"
        android:onClick="sendMessage" />
</LinearLayout>
```


The `android:onClick` attribute's value, "sendMessage", is the name of a method in your activity that the system calls when the user clicks the button.

MainActivity.java

```
public final static String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
}

/** Called when the user clicks the Send button */
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```



An Intent is an object that provides runtime binding between separate components (such as two activities). The Intent represents an app's "intent to do something." You can use intents for a wide variety of tasks, but most often they're used to start another activity.

The constructor used here takes two parameters:

A Context as its first parameter (this is used because the Activity class is a subclass of Context)

The Class of the app component to which the system should deliver the Intent (in this case, the activity that should be started)

An intent not only allows you to start another activity, but it can carry a bundle of data to the activity as well. An Intent can carry a collection of various data types as key-value pairs called extras. The putExtra() method takes the key name in the first parameter and the value in the second parameter.

DisplayMessageActivity.java

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Get the message from the intent
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

    // Create the text view
    TextView textView = new TextView(this);
    textView.setTextSize(40);
    textView.setText(message);

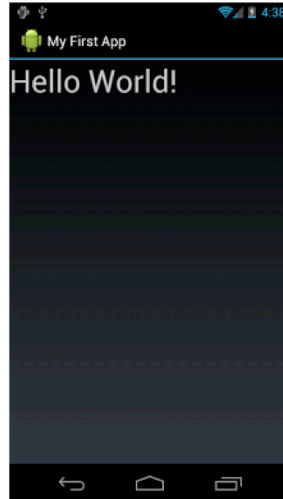
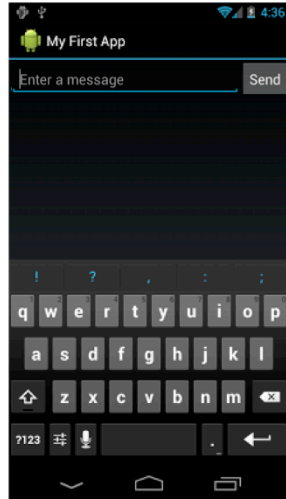
    // Set the text view as the activity layout
    setContentView(textView);
}
```

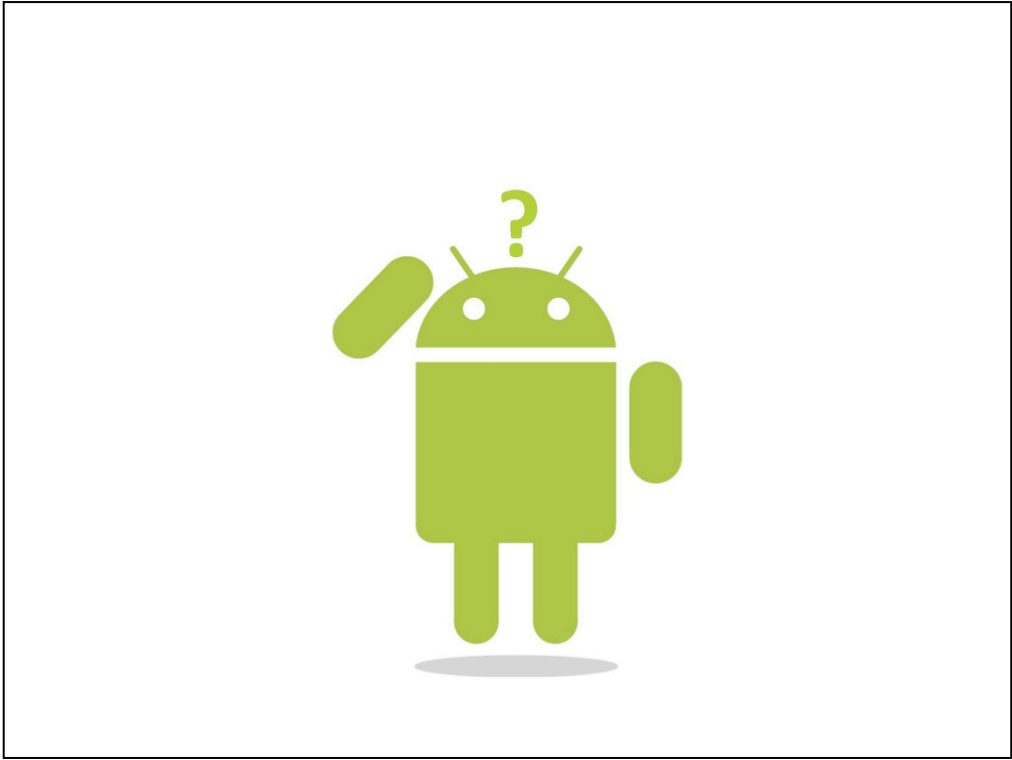
Every Activity is invoked by an Intent, regardless of how the user navigated there. You can get the Intent that started your activity by calling `getIntent()` and retrieve the data contained within it.

To show the message on the screen, create a `TextView` widget and set the text using `setText()`. Then add the `TextView` as the root view of the activity's layout by passing it to `setContentView()`.

This is the procedural way of creating the view and its elements. Using the resource file, as shown in Main Activity was an example of declarative way.

This is how it looks





Remember

Register for groups in Bilda

Lab 1 deliverable by **2nd Feb**

