Unit 3:
Android SDK and Basic Java for Android Apps

Mobile Application Development (MAD)
CCIT4059, 2025-2026

Outline

- Android SDK
  - Activity and Its Java Class
- Basic Java for MAD
  - Typical Structure of Java Class
- Java Classes and Objects
  - Constructors / Methods

** This unit briefly introduces and summarises Java Programming Language for developing Android apps. Students should refer to other materials for details if necessary.*

Software Development Kit

Software Development Kit (SDK):

- Major mobile OSs provide SDK, associated with major supporting programming languages and development environment, for 3rd parties to develop apps running on them.

Key functions:

- Tools for building mobile apps
- Libraries, compilers, emulators, documentation
- Accessing device features
- Ensuring compatibility with the mobile OS

ANDROID SDK

Core Development Tools

- Android Studio is the official IDE provided by Google providing comprehensive and essential tools like code editing, debugging, and performance profiling.
  - It includes essential components such as libraries, a debugger, an emulator
  - It provides a graphical interface that consolidates various tools, making the development process more efficient and user-friendly

Programming Languages Supported

- The SDK supports Java and Kotlin, allowing developers to choose based on project needs and preferences.

Android SDK

Build and Automation

- Gradle automates the build process, enabling modular and efficient app packaging.

UI Frameworks and Libraries

- Android uses XML layouts and Jetpack Compose for declarative and reactive UI development.
  - *Jetpack Compose is Android's modern toolkit for building native UIs, simp lifying UI development with less code and powerful tools.*

- Remark: Google regularly releases a new version of Android, a corresponding SDK is also released, ensuring that developers have access to the latest features and tools.

IOS SDK (APPLE)

For reference

Development Tools and Environment

- Xcode integrates coding, UI design, debugging, testing, and asset management in one interface for efficient app development.

Programming Languages

- Developers primarily use Swift for safe, clear, and performant code, with Objective-C still supporting legacy apps.

UI Frameworks

- UIKit and SwiftUI frameworks enable building adaptive user interfaces for multiple Apple devices with ease.

Advanced SDK Features

- Frameworks like Metal, Core ML, and ARKit support graphics, machine learning, augmented reality, and security.

HarmonyOS SDK

For reference

Distributed Multi-Device Capability

- HarmonyOS SDK enables applications to run seamlessly across mobile, wearable, smart home, and IoT devices collaboratively.

Development Languages and IDE

- Developers use Java, JavaScript, and ArkTS with DevEco Studio IDE tailored for HarmonyOS app creation.

Ability Framework and Modular Design

- Ability framework supports modular app components like UIAbility for interfaces and ServiceAbility for background tasks.

Super Device Features

- HarmonyOS enables interaction with multiple devices as one, enhancing user workflows and device integration.

Key Components of Android SDK

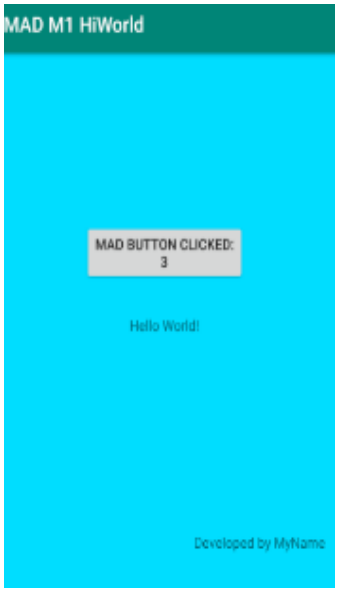
1. SDK Platforms
   - Each Android version provides a corresponding SDK Platform that **includes the Android framework APIs and system libraries f**or compiling apps.
2. Android SDK Platform-tools
   - A collection of device-level tools (e.g. ADB and Fastboot), used for **debugging, flashing, and communicating** with devices.
3. Gradle and Android Gradle Plugin (AGP)
   - **Tools to configure, and automate the entire build process**. Gradle (Build engine) handles all the build steps, while AGP (plugin) tells Gradle how an Android app should be compiled, packaged, and prepared for running.
4. Android SDK Build-Tools
   - **Tools required to build Android app binaries (APK/AAB)**
5. Android Emulator
   - A virtual Android device that lets you **run and test apps without needing physical hardware.**

Android SDK
(Activity and Its Java Class)

- Android apps (applications) are mostly created with component known as Activity

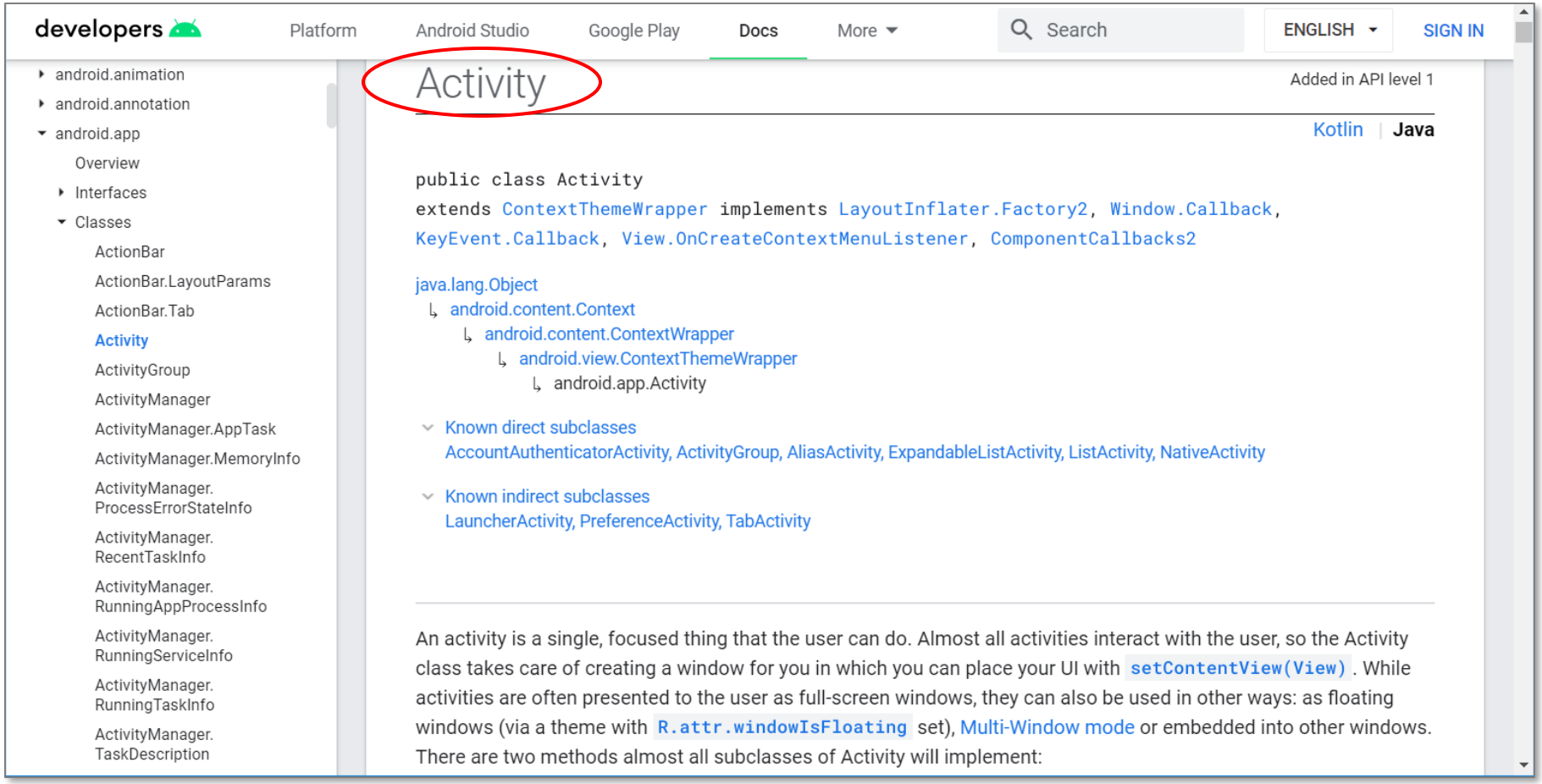
- An activity is a single, standalone module of application functionality that usually correlates directly to a single user interface screen and its corresponding functionality

- Activity is essentially a Java class / object
  - As such, learning Java programming language is one important way to start implementing Android apps



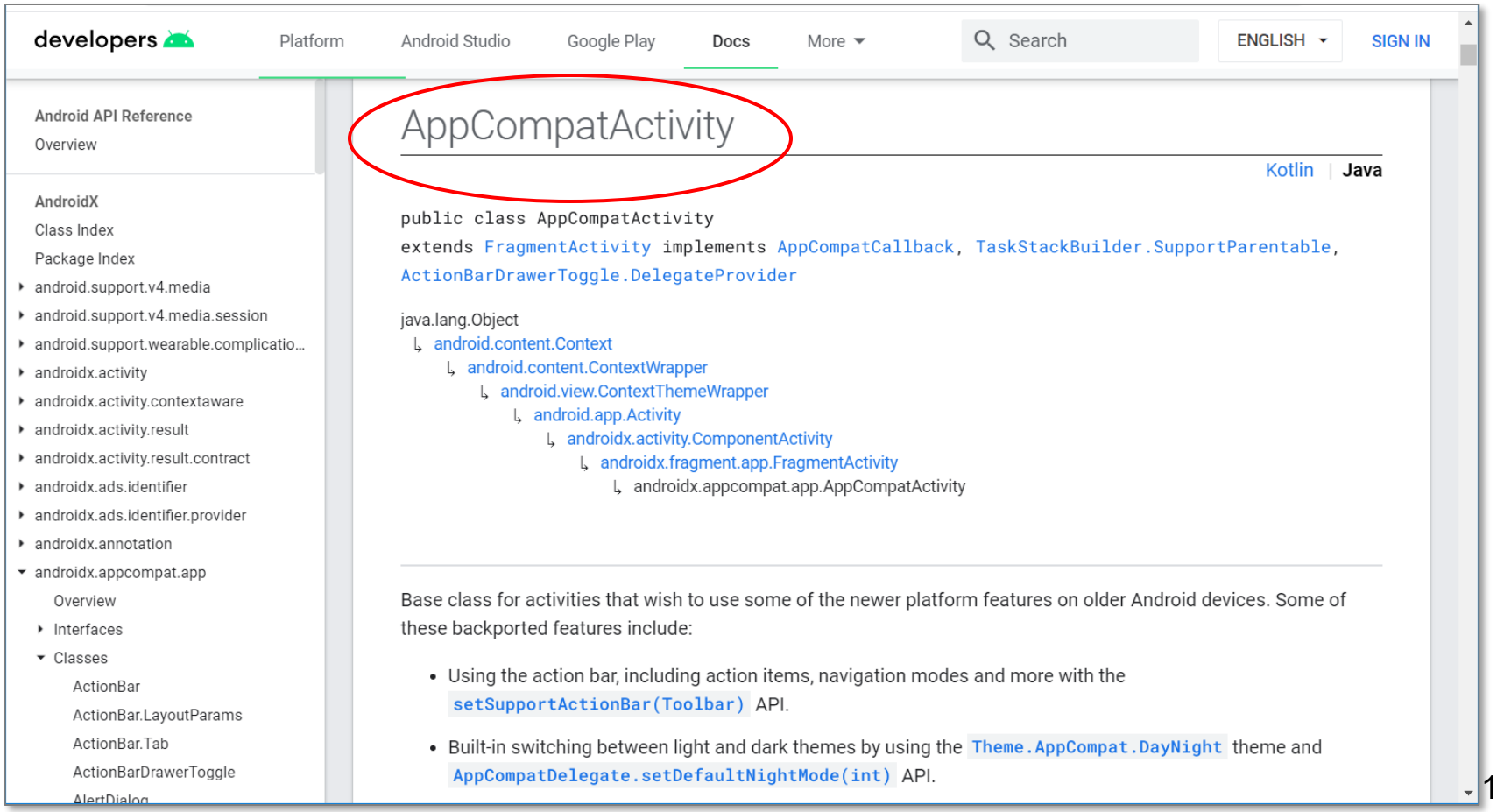
Android SDK
(Activity and Its Java Class)

- For details of a Java class, we need to refer to API (Application programming interface) information



Android SDK
(Activity and Its Java Class)

- Often Android app is built on a subclass of `Activity`, `AppCompatActivity`
- Here shows its API information



Basic Java for MAD

Self-learning

- Java is a class-based programming language that supports object-oriented (OO)

- Java is originally intended to let developers "write once, run anywhere" (WORA)
  - Java compiled code (bytecode, the `.class` file) that runs on one platform does not need to be recompiled to run on another

- Java is designed with a C/C++ style syntax that many programmers would find familiar
  - As such people with C/C++ background may find learning Java easier than others

Basic Java for MAD

Self-learning

- Java is designed with similar ***syntax of C++ / C style***
- **Data types** : Java programs supports classes and their objects (OO)
  - Java also supports primitive types (e.g. `int`, `float`, `boolean`)
- **Blocks of codes** `{}` : Java uses blocks with brace pair `{}`, e.g. for compound statements, class body, and method body
  - Unlike Python, ***indentation in code lines will be ignored in Java***
- **Declaration** of variables (with data type) : Java variables must be first declared (with specified data type), before accessing
  - `int aNum; // variable (named aNum) declared as int type`
- **Statement end** `;` : Statements end with a semi-colon symbol (`;`), e.g.

```
aNum = 123; // an assignment statement
```

Basic Java (Reserved keywords)

Self-learning

- 51-character sequences are reserved for use as keywords and cannot be used as identifiers
  - There are extra 10 “restricted keywords” newly introduced for “module” declarations: `exports`, `module`, `open`, `opens`, `provides`, `requires`, `uses`, `with`, `to` and `transitive`. They may be used as identifiers anywhere else in the code.

abstract	**continue**	**for**	**new**	**switch**
assert***	**default**	**goto***	**package**	**synchronized**
boolean	**do**	**if**	**private**	**this**
break	**double**	**implements**	**protected**	**throw**
byte	**else**	**import**	**public**	**throws**
case	**enum******	**instanceof**	**return**	**transient**
catch	**extends**	**int**	**short**	**try**
char	**final**	**interface**	**static**	**void**
class	**finally**	**long**	**strictfp****	**volatile**
const*	**float**	**native**	**super**	**while**
_ (underscore)				

* *not used;*
** *added in 1.2*
*** *added in 1.4*
**** *added in 5.0*

- **true** and **false** are not keywords, but rather boolean literals.
- **null** is not a keyword, but rather the null literal.
- **var** is not a keyword, but rather an identifier with special meaning as the type of a local variable declaration (from version 10).

Basic Java for MAD

Self-learning

- Java is also referred to a software platform
  - Java platform has two major components:
    - Java Runtime Environment (JRE) (e.g. Java Virtual Machine)
      - Purpose: For developing Java applications
    - Java Development Kit (JDK) (e.g. javac (Java compiler), Debugging tools)
      - Purpose: For running Java applications
- Three major official editions of Java Platform:
  - Java SE: Core Java platform for general-purpose apps
  - Java EE: Enterprise-grade distributed applications
  - Java ME: Feature-phone/embedded/IoT devices (devices with limited CPU, RAM, and storage) (Nokia, Sony-Ericsson, Motorola phone)
- **But Android apps built with Java are running on the specific Android platform.** (Android adopts Android Runtime (ART))

Android Runtime (ART) and Java Virtual Machine (JVM)

What is Android Runtime (ART)?

- ART is the system in Android that runs apps.
  - Every Android app is written in Java or Kotlin, but the phone cannot directly understand this code.
  - ART acts like a translator that converts the app's code into machine code the device can understand.

- Function of ART
  - Converts app code into native machine code
  - Speeds up app launching
  - Helps with background tasks

Android Runtime (ART) and Java Virtual Machine (JVM) (How ART Works in Android Apps (Workflow))

- When developers write an Android app (in Java or Kotlin):
  - The code is compiled into **.class** files
  - Then it becomes **.dex** (Dalvik Executable) files
  - During installation, ART pre-compiles the .dex into **native machine code**
  - When the user taps the app, it launches quickly because the code is **already compiled**

- ART compiles the app **before** you use it
  - It's like preparing all the ingredients before cooking—the cooking becomes fast and smooth

Android Runtime (ART) and Java Virtual Machine (JVM)

The difference between ART and JVM

Android Runtime (ART)

- Built specifically for Android devices.
- Executes DEX (Dalvik Executable) bytecode optimized for mobile devices, not Java bytecode.

Java Virtual Machine (JVM)

- A general-purpose virtual machine for running Java applications on desktops, servers, and many platforms.
- Executes Java bytecode (.class) directly and runs standard Java bytecode produced by javac.

Android Runtime (ART) and Java Virtual Machine (JVM)

Java normally uses the **Java Virtual Machine (JVM) with** a different workflow.

Android (ART): Ahead-Of-Time (AOT) Compilation

- Code is compiled **before the app runs** (during installation)
- Faster app start-up and lower memory usage
- More efficient on mobile hardware

Java (JVM): Just-In-Time (JIT) Compilation

- Code is compiled **while the program is running**
- The JVM translates bytecode *on the fly*
- Slower start-up speed and higher in memory usage
- But flexible and easy to run on many platforms

Basic Java for MAD:(Create an app with Java)

Workflow to create an Android App with Java

1. Setup
   - Install Android Studio (includes JDK + SDK).
   - Create a New Project → choose Empty Activity.
   - Android Studio generates basic files automatically (Java class, XML layout, manifest

2. Code
   - Write Java code in MainActivity.java.
   - Design UI in XML layout files (e.g., activity_main.xml).
   - Add buttons, text fields, and logic to respond to user actions.

3. Compile
   - Compile Java into Java bytecode
   - Convert bytecode into DEX format
   - Package the project into an APK

4. RUN
   - Choose a device: Android Emulator (Virtual Device), or Physical Android phone (USB debugging enabled)

Basic Java for MAD
(Create an app with Java: - setup, code, compile, run)

1. SETUP development environment,

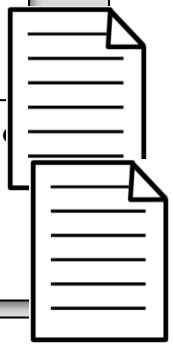
for specific apps on specific platform

2. CODE source code *.java file,

with APIs for apps on specific platform

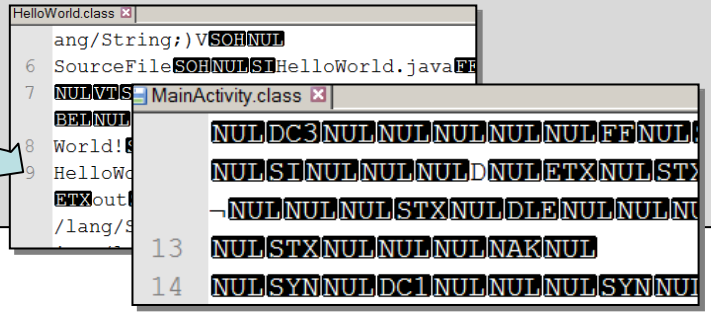
```
class HiWorld extends SuperHi
{
 ... "Hi\nWorld!"); // Hi mesg...
```

```
class MoreHiWorld extends HiWorl
{
  ...
}
```



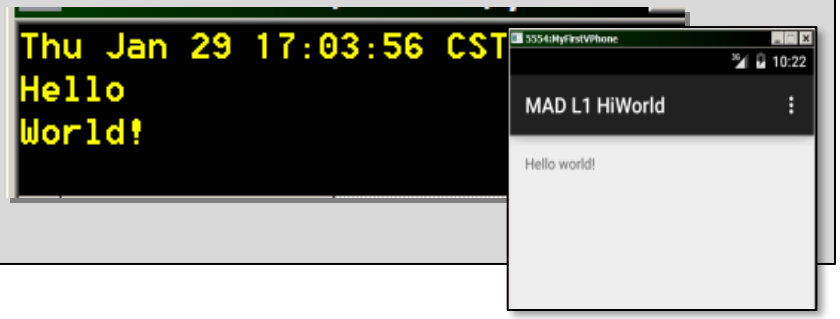
3. COMPILE (e.g. *.java),

*into a bytecode file (e.g. *.class file)*



4. RUN, *with INTEGRATING*

e.g. other modules if needed (including other sources, e.g. XML, images, etc.)



Basic Java for MAD

- An app including modules developed with Java code essentially is composed of Java classes and their objects / instances

- Each Java class has a typical structure of:
  - `package` statement (if any), at the top
  - `import` statements (if any),
  - `class` declaration, often includes
    - Fields (Data members), similar to internal (C) variables
    - Constructors, similar to special methods
    - Methods, similar to internal (C) functions of object
  - Comments (if any); could be anywhere

Basic Java for MAD
("PureJava" Example: Class `PureJavaActivity`)

Reference Only

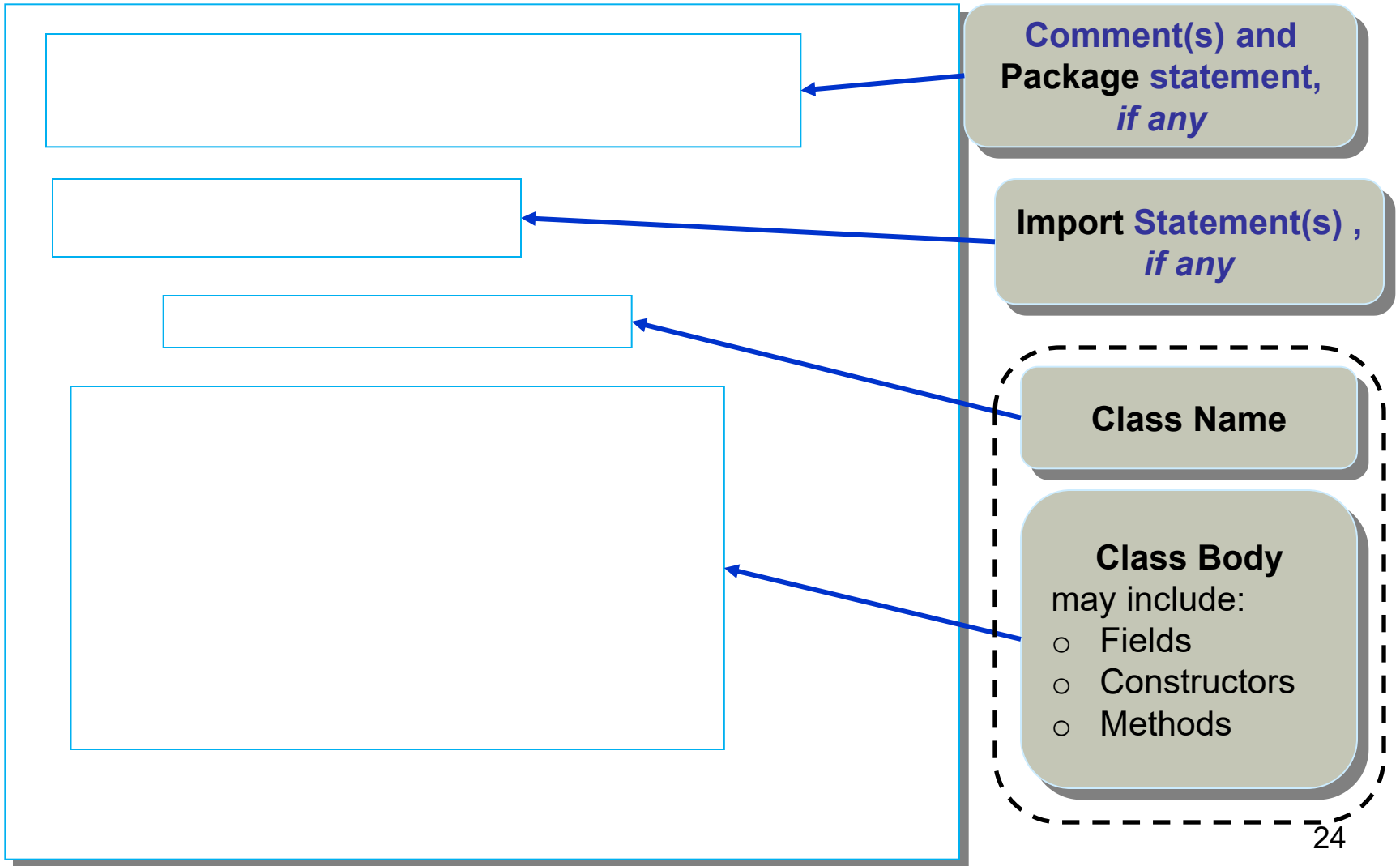
```
package mad.m3purejava;     // package statement
import android.os.Bundle;   // import statement
// ... More codes here, for proper import statements here
public class PureJavaActivity extends AppCompatActivity {   // a Java class declared/defined
    RelativeLayout aRLayout = null;     // field for UI layout, like a layout XML
    Bicycle bikeA;                      // a field, Bicycle
    Button aButA;                // a field, Button: UI component
    @Override // for overriding superclass's method
    protected void onCreate(Bundle savedInstanceState) { // app starts / creates
        super.onCreate(savedInstanceState); // always call superclass
 // Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A");
        bikeA.setSpeed(123); // access object, outside its class

        aButA = new Button(this);    // new Button for Bike A, to be added to layout
        aButA.setBackgroundResource(R.drawable.ic_baseline_directions_bike_24);
        aButA.setTextColor(Color.parseColor("#FF0000"));  // set color, say red
        aButA.setTextSize(20);
        aButA.setText(bikeA.name);    // set Text of Button, name of Bike
// set an OnClickListener object (Lambda Expression) as button's listener to handle event
        aButA.setOnClickListener(v-> { //method called if  source view (Button) is clicked
                aButA.setX(aButA.getX() + bikeA.getSpeed()); // move
                if (aButA.getX() + aButA.getWidth() > aRLayout.getWidth())
                    aButA.setX(0);
        });

        aRLayout = new RelativeLayout(this);      // new (create) the layout
        aRLayout.setBackgroundColor(Color.YELLOW);        // set bg color of the layout
        aRLayout.addView(aButA);      // add the button created earlier to layout
        aButA.setX(100);  // set the initial x position of button A (Bike A)
        aButA.setY(100);  // set the initial y position
        setContentView(aRLayout); // set the layout to be the activity window (content view)
    }
}
```

Basic Java for MAD
(Typical Structure)

- Typical form in a Java file, defining a class



Basic Java for MAD (Typical Structure)

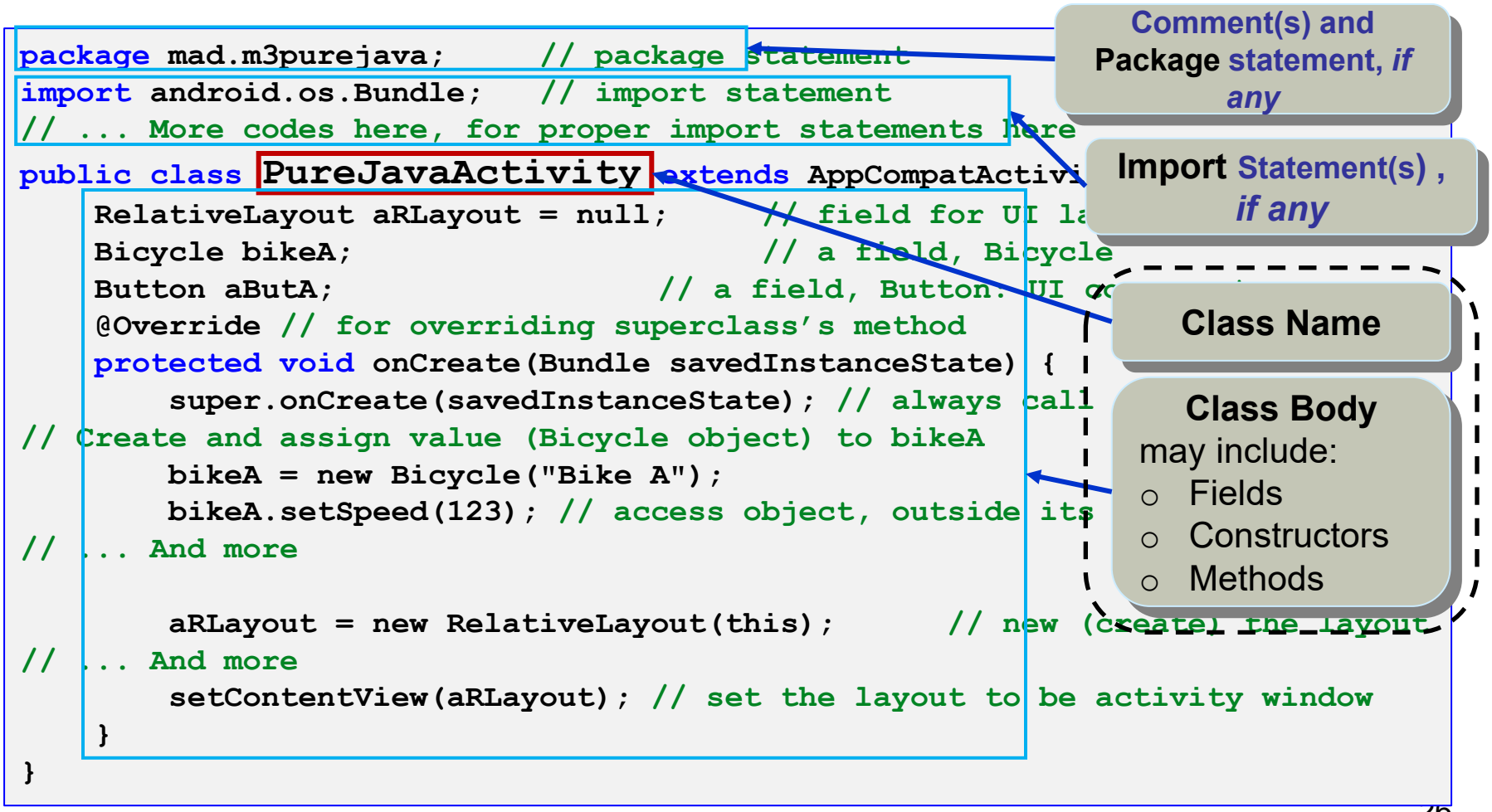
- E.g. The following declares (defines) a Java class named **`PureJavaActivity`**, with some fields (for attributes) and a method `onCreate` (for behavior) in the class body

```
package mad.m3purejava;     // package statement
import android.os.Bundle;   // import statement
// ... More codes here, for proper import statements here
public class PureJavaActivity extends AppCompatActivity {// Declare a class
    RelativeLayout aRLayout = null;     // field for UI layout
    Bicycle bikeA;                      // a field, Bicycle
    Button aButA;                   // a field, Button: UI component
    @Override // for overriding superclass's method
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // always call superclass
// Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A");
        bikeA.setSpeed(123); // access object, outside its class
// ... And more

        aRLayout = new RelativeLayout(this);      // new (create) the layout
// ... And more
        setContentView(aRLayout); // set the layout to be activity window
    }
}
```

Basic Java for MAD (Typical Structure)

- Corresponding to the typical form with this Java **`PureJavaActivity`** and its structure



Basic Java for MAD
(Basic Syntaxes with `PureJavaActivity` Example)

- Class

```
<modifier(s)> class <class name> {
    <class body>
}
```

- Field (Similar to variable)

```
<modifier(s)> <data type> <field name>
```

- Constructor

```
<modifier(s)> <class name>
(<parameter(s)>){
        <constructor body>
}
```

- Method

```
<modifier(s)> <return type>
<method name> (<parameter(s)>){
        <method body>
}
```

- Class

```
class PureJavaActivity extends
AppCompatActivity {
    <class body>  }
```

- Field

```
RelativeLayout aRLayout = null;
Bicycle bikeA;
Button aButA;
```

- Constructor

** No Specified Constructor*

- Method

```
protected void onCreate
(Bundle savedInstanceState){
        <method body>
}
```

Basic Java for MAD
(Package and Import Statements in Android Apps)

- Package and Naming
  - A package in Java is a bundle or library of classes and interfaces (or related types)
  - Each package has a name written in all lower case to avoid conflict with the names of classes or interfaces
  - Often companies use their reversed Internet domain name to begin their package names – e.g. `com.abc.mypackage` for a package named `mypackage` created by a programmer at the company with domain name `abc.com`.

Basic Java for MAD
(Package and Import Statements in Android Apps)

- `package` statement
  - To make your class(es) (in a source file) as a member of the package, put the package statement with package name at the top of source file
    - *This could be done with the help of IDE (e.g. Android Studio)*
  - In the `PureJavaActivity` sample code, the package statement below is the first statement (at the top of the source file), and indicates this class (`PureJavaActivity`) belongs to the package `mad.m3purejava`

```
package mad.m3purejava;
```

Basic Java for MAD
(Package and Import Statements in Android Apps)

- Using a public package class from outside its own package, there are three ways:
  1) Import the package member (similar to include a library in C)
  2) Import the member's entire package
  3) Refer to the member by its fully qualified name

- `import` statement is put at the beginning of the file before any type definitions, but after the package statement if any
  - *This could also be done with the help of IDE (e.g. Android Studio)*

Basic Java for MAD
(Package and Import Statements in Android Apps)

- In our `PureJavaActivity` example, 3 ways to use the `Bundle` class in `android.os` package

1) Import the package specific member:

```
import android.os.Bundle; // import one specific member
// ... Inside the class body
  protected void onCreate(Bundle savedInstanceState){ //original, import statement required
```

2) Import the member's entire package (includes all members)

```
import android.os.*;// import entire package, including Bundle
// ... Inside the class body
  protected void onCreate(Bundle savedInstanceState){ //original, import statement required
```

3) Refer to member by its fully qualified name (***without*** **`import`** statement needed):

```
// ... Inside the class body
   protected void onCreate(android.os.Bundle savedInstanceState){
```

Basic Java

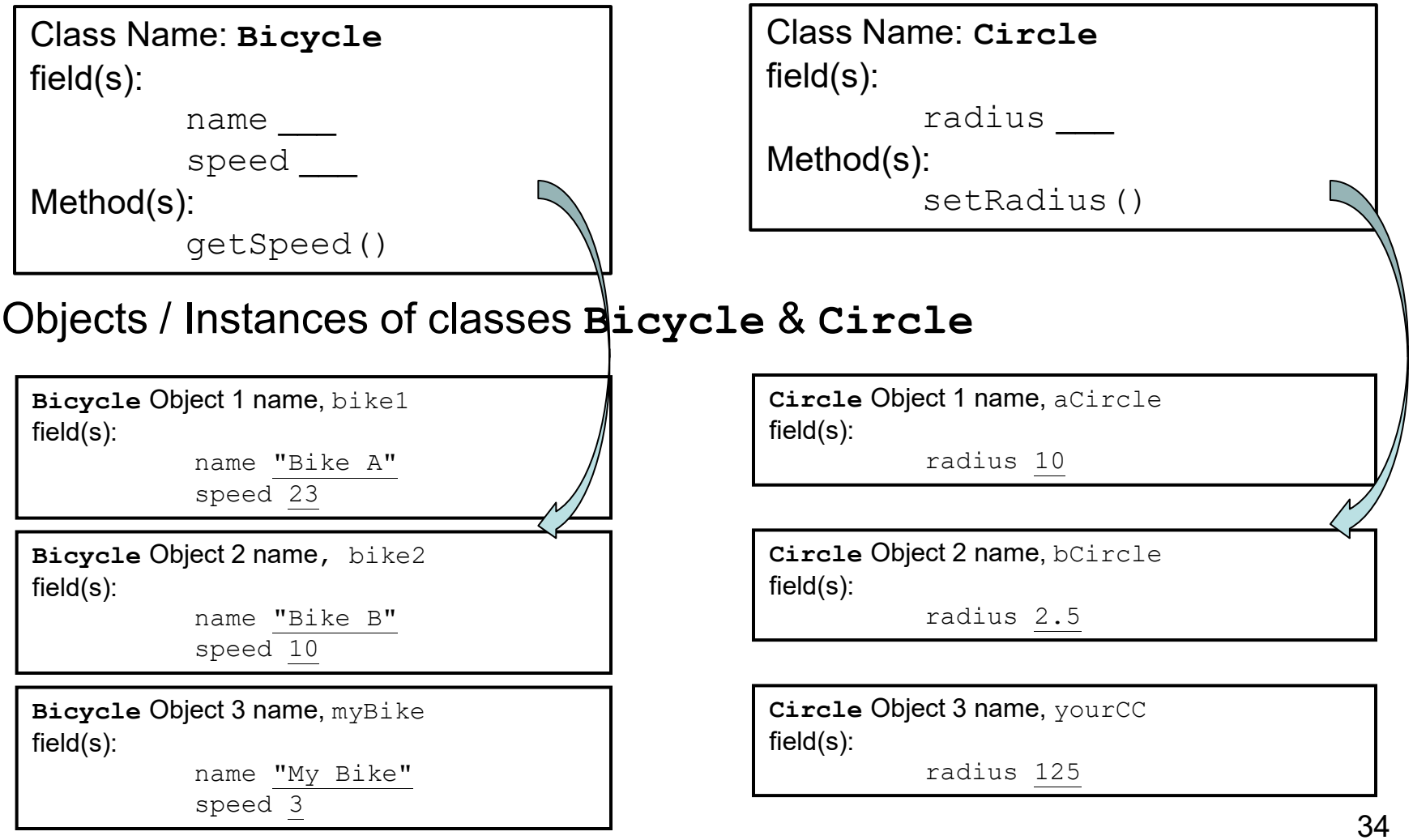
- Basic Java class and object-oriented features:
  - Defining and Using **classes** and their **objects**
    - Classes / Objects are associated with both attributes and behaviors (fields and methods in Java)
  - The major 3 object-oriented features are:
    - **Inheritance** relationship
      - A mechanism designed for different entities (subclasses) that share common features (from superclass)
    - **Encapsulation** (information hiding)
      - Internal components are encapsulated from outside
    - **Polymorphism** to morph into many different forms
      - E.g. Overloading and overriding methods(with same name)

Java Classes and Objects

- Classes and their Objects
  - Objects represent entities (e.g. `bike1`, `bike2`, `aCircle`) with attributes/fields (`speed`, `radius`) and behaviors/methods (`getSpeed()`, `setRadius()`)
  - Objects of same type defined using the same class
    - E.g. Objects `bike1` and `bike2` of class `Bicycle`
  - A class is a type, template or blueprint that defines what an object's attributes/properties (***fields***, or data members) and behaviors (***methods***) will be
  - An object is an *instance* of a class, and many objects/ instances can be created from a class
    - Terms "object" & "instance" are often interchangeable

Java Classes and Objects
(Example Classes and Their Objects)

2 Classes:- **`Bicycle`** (left) & **`Circle`** (right):



Java Classes and Objects (Example Classes and Their Objects)

- Following declares a Java class **`Bicycle`**, with field, constructor and method

```
// ... package and import statements here, if necessary
public class Bicycle { // class Bicycle declared here

    // Field (instance variable)
    String name; // name of  the bicycle object
    private int speed; // speed of  the bicycle object

    // Constructor: Initializes the field
    public Bicycle(String inName) {
        name = inName;
        speed = 10; // for initial value
    }

    // Method, get/return the speed of this bicycle
    public int getSpeed( ) {
        return speed;
    }

    // Method, set/assign the speed of this bicycle
    public void setSpeed(int newSpeed) {
        speed = newSpeed;
    }
}
```

Java Classes and Objects
(Example Classes and Their Objects)

- Below declares a class **`BicycleSpeedComp`**, a main class using class **`Bicycle`**

```
// ... package and import statements here, if necessary
public class BicycleSpeedComp {
// Java SE Program starts below, the main() method
    public static void main(String[] args) {
        Bicycle bikeA, bikeB;
        int  speedA, speedB;
        // Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A");
        bikeA.setSpeed(23); // access object, outside its class
        // Create and assign value (Bicycle object) to bikeA
        bikeB = new Bicycle("Bike B");
        // Output or Display the information
        speedA = bikeA.getSpeed( );
        speedB = bikeB.getSpeed( );
        if (speedA > speedB)
            System.out.println("Bike A is faster than Bike B.");
    }
}
```

Java Classes and Objects (Using a Class / Object)

- Although Java programs require designing and implementing Java classes, it is even more important to know how to *use, create and access objects of existing classes* (e.g. those defined in Android APIs) for starting to make simple mobile apps

- Typical steps of using a class and its object:
  1) ***Declare*** an object of a specific class
  2) ***Create*** (reserve space & initialize), often with ***assign***
  3) ***Access*** (get/set attributes and call methods)

Java Classes and Objects (Using a Class / Object)

- ***Declare*** object of a class: Syntax

  <ClassName> <ObjectName> e.g. `Button aButA`

- ***Create*** / *Instantiate* an object: Syntax, with a keyword `new`

  new <class name>(<arguments>) e.g. `new Button()`

```
public class PureJavaActivity extends AppCompatActivity {
    RelativeLayout aRLayout = null;     // field for UI layout
    Bicycle bikeA;     // Declare a field, object of Bicycle
    Button aButA;   // Declare a field object, Button: UI component
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // always call superclass
// Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A"); // Create object, & assign
        bikeA.setSpeed(123); // access object, outside its class
// ...
}
```

Java Classes and Objects (Using a Class / Object)

Example: Use the **`Bicycle`** *class as example:*

1) ***Declare*** (tell there is an object, with name)

```
   Bicycle bikeA;     // Declare an object, of Bicycle
// An object name bikeA of class(type) Bicycle
```

2) ***Create*** (reserve space and initialize), & *assign*

```
   bikeA = new Bicycle("Bike A"); // create with new
// Bicycle has a constructor will be called:
// public Bicycle(String inName)
```

3) ***Access*** (call a method in the example below)

```
   bikeA.setSpeed(123); // access object, method calling
```

Java Classes and Objects (Using a Class / Object)

Example: Use the **`String`** *class as another example:*

1) ***Declare*** (tell there is an object, with name)

```
String aStr; // declare a String
```

2) ***Create*** (reserve space and initialize), & *assign*

```
aStr = new String("A Mesg"); // create and assign;
```

3) ***Access*** (call a method in the example below)

```
char aCh = aStr.charAt(2);
// charAt is a method of String, which returns the character //
according to the argument as the index (2 in this case)
// charAt acts like a C's function for object aStr;
// And thus aStr.charAt behaves like aStr's charAt function
// After execution, aCh above is assigned with 'M' ;
```

Java Classes and Objects (`String`, an Important Java Class)

- A `String` object in Java is *immutable* (i.e. its content cannot be changed once constructed)

- `String` is a special and important Java class that its object string can be constructed in 2 ways:

  1) Using the **`new`** operator, e.g.

  ```
  String aStr = new String("A new string");
  ```

  2) Directly assigning a string literal, e.g.

  ```
  String bStr = "A literal string";
  ```

- One popular string operation is concatenation, the `+` operator, e.g.

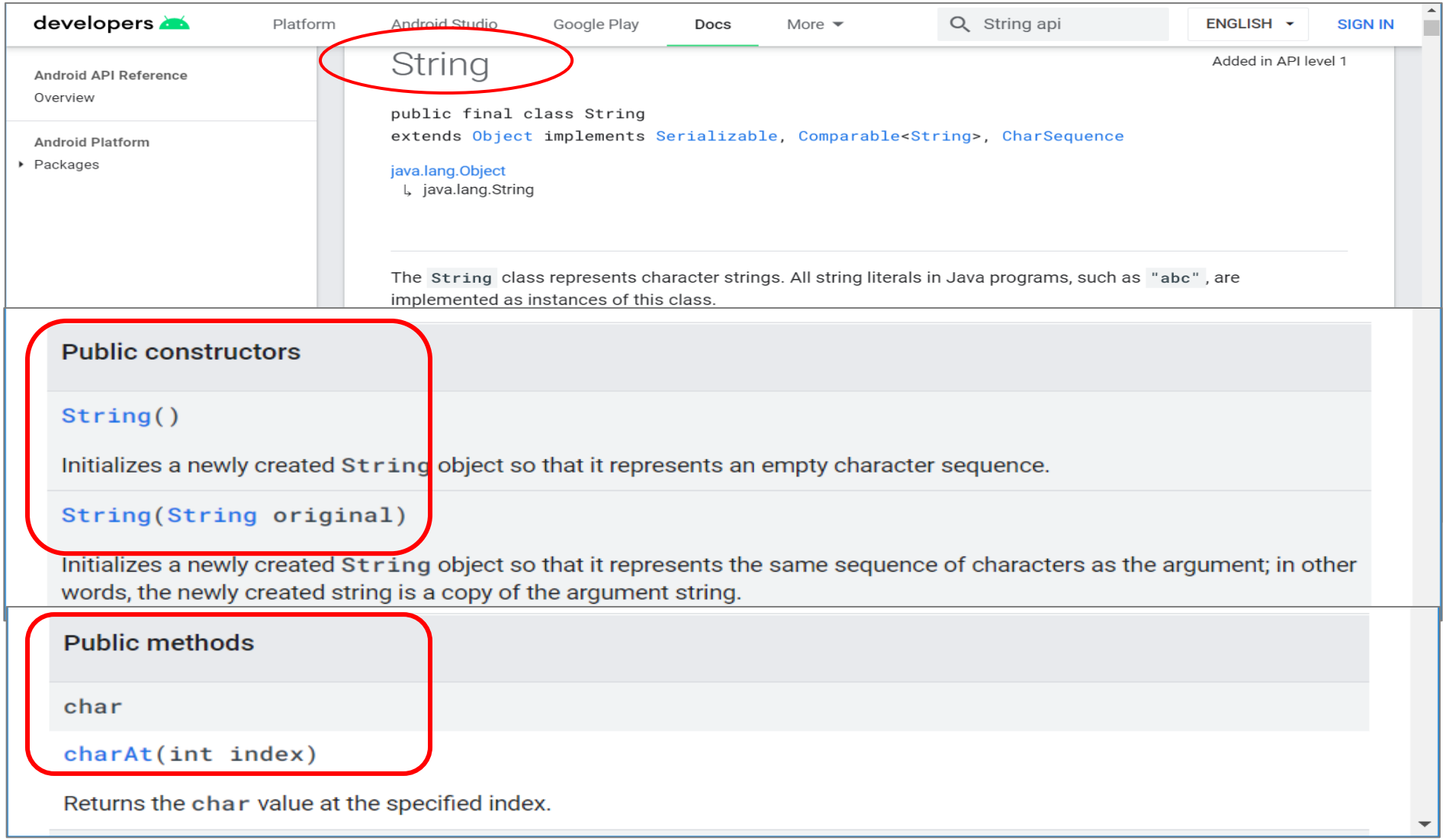
```
output = "Great " + "MAD";
 // output "Great MAD"
```

```
output = "Class: " + "CL" + 3;
 // output "Class: CL3"
```

Java Classes and Objects (Using a Class / Object)

Reference Only

- For reference, **`String`** API document:



Java Classes and Objects (Declare a Class)

- Often we have to define and declare our own class, for specific task
  - Class declaration header line: Syntax, with a keyword `class`

    `<Modifier(s), optional> class <ClassName>`

```
public class PureJavaActivity extends AppCompatActivity {
    RelativeLayout aRLayout = null;     // field for UI layout
    Bicycle bikeA;    // Declare a field, object of Bicycle
    Button aButA;   // Declare a field object, Button: UI component
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // always call superclass
// Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A"); // Create object, & assign
        bikeA.setSpeed(123); // access object, outside its class
// ...
}
```

Java Classes and Objects (Fields in a Class)

- Field (declare and access)
  - Like a (C) variable associated to a specific object
  - A field can be accessed within the class, for assigning & getting values (e.g. below)

** Use dot-notation if accessing them outside the class*

```
public class PureJavaActivity extends AppCompatActivity {
    RelativeLayout aRLayout = null;  // field for UI layout
    Bicycle bikeA; // Declare a field, object of Bicycle
    Button aButA;  // Declare a field object, of Button
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState); // always call superclass
// Create and assign value (Bicycle object) to bikeA
       bikeA = new Bicycle("Bike A"); // Create object, & assign
       bikeA.setSpeed(123); // access object, outside its class
// ...
}
```

Java Classes and Objects (Constructors in a Class)

- Constructor (*Must use the class name*)
  - Like a special method, without return type
  - Mainly for initializing objects
  - Called when creating object with keyword **`new`**
  - Below **`Bicycle`** class has a one-argument constructor

```
// the Bicycle example Java class
// ... package and import statements here
public class Bicycle { // class Bicycle
 // Field (instance variable)
     String name; // name of bicycle object
    private int speed; // speed of bicycle

// Constructor: Initializes the field
    public Bicycle(String inName) {
          name = inName;
          speed = 10; // for initial value
    }
    // Methods ...
// ...
}
```

```
// ... package and import statements here
public class BicycleSpeedComp {
// Java SE Program starts at main() method
    public static void main(String[] args) {
          Bicycle bike1, bike2;
          int  speed1, speed2;
// Create and assign objects to bike1
          bike1 = new Bicycle("Bike A");
          bike1.setSpeed(35); // access object
// Create and assign objects to bike2
          bike2 = new Bicycle("Bike B");
// ...
    }
}
```

Java Classes and Objects (Methods in a Class)

- Method (define and access)
  - Like a (C) function associated to a specific object
  - To call an object method (similar to call a C's function), we may use dot-notation (as below)

    <object>.<methodname>(<parameter(s)>)

** Dot-notation may be omitted if accessing them inside class*

```
public class PureJavaActivity extends AppCompatActivity {
    RelativeLayout aRLayout = null;  // field for UI layout
    Bicycle bikeA; // Declare a field, object of Bicycle
    Button aButA;  // Declare a field object, of Button
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // always call superclass
// Create and assign value (Bicycle object) to bikeA
        bikeA = new Bicycle("Bike A"); // Create object, & assign
        bikeA.setSpeed(123); // access object method, outside class
// ...
}
```

Java Classes and Objects (Overloading Constructors / Methods)

- **Overloading** Constructors / Methods: two or more methods / constructors have the same name inside a class, based on rules (below) to make them distinguishable:
- They have ***different numbers of parameters***, e.g.

```
MADClass(); // overloading constructors

MADClass(int abc);
```

- Their parameters are of ***different data types*** when the number of parameters is the same, e.g.

```
void setFe(float abc); // overloading methods

void setFe(String abc);
```

** This is closely related to the concept of polymorphism (Removed in Unit 3)*

References

- Part of this slide set is prepared and extracted from the followings:
  - Friesen, J. (2014) *'Learn Java for Android Development'*, 3ed, Apress Media
  - Gargenta, M. and Nakamura, M. (2014) *'Learning Android: Develop Mobile Apps Using Java and Eclipse'*, 2ed, O'Reilly Media, Inc.
  - Gok, N. and Khanna, N. (2013) *'Building Hybrid Android Apps with Java and JavaScript'*, O'Reilly Media, Inc.
  - Nolan, G., Cinar, O. and Truxall, D. (2014) *'Android Best Practices'*, Apress Media
  - Smyth, N. (2014) 'Android Studio Development Essentials', CreateSpace (Web: http://www.techotopia.com/index.php/Android_Studio_Development_Essentials)
  - Wolfson, M. (2013) *'Android Developer Tools Essentials'*, O'Reilly Media, Inc.
  - Google Inc. 'Website for Android Developers', http://developer.android.com/
  - Google Inc. 'Website for Android Sources', https://source.android.com/
  - Apple Inc. 'Website for Apple Developers', Website: https://developer.apple.com/
  - Microsoft Corp. 'Website for Windows Developers', https://developer.microsoft.com/en-us/windows
  - Wikipedia Website: http://en.wikipedia.org
- This set of slides is for teaching purpose only