Android Application Design

Software System Design

Zhu Hongjun

Session 5: Programming

- Structured & OO Programming
- Programming for Performance
- File Programming
- Network Programming
- Multi-media Programming
- Conclusions

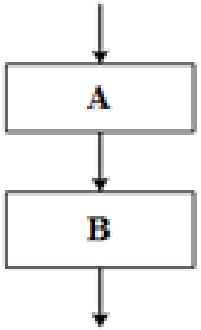
Structured & OO Programming

- Structured programming
  - a programming paradigm aimed at improving the clarity, quality, and development time of a computer program
  - by making extensive use of subroutines, block structures and for and while loops
  - all programs are seen as composed of three control structures
    - Sequence, Selection, Iteration

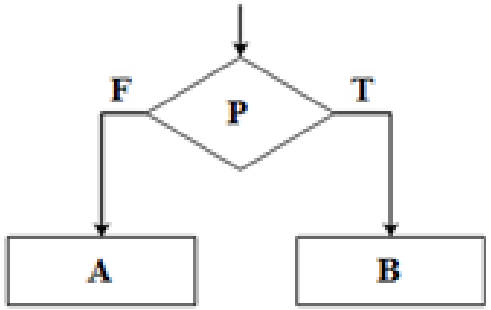
Structured & OO Programming

- Program flowchart
  - a program flowchart depicts visually via symbols and lines the logic flow of a program and the interactions it performs
  - 5 basic control structures can be applied to depict structured programs
    - sequence, selection, case, do-while, do-until
  - you can deploy structured design ideas by some diagramming tools

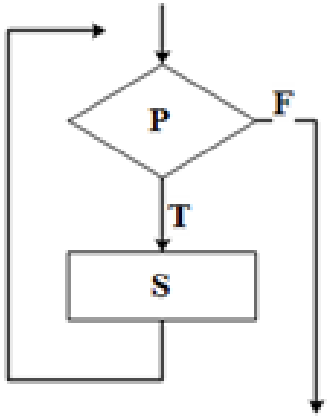
Basic Control Structure of Program Flowchart



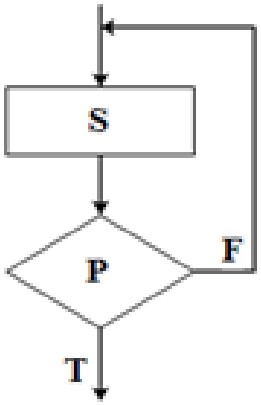
(1). SEQUENCE



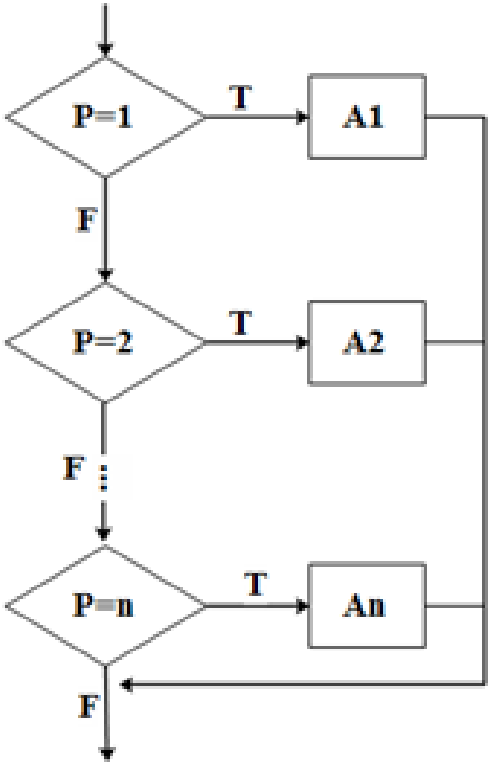
(2). SELECTION



(3). DO-WHILE

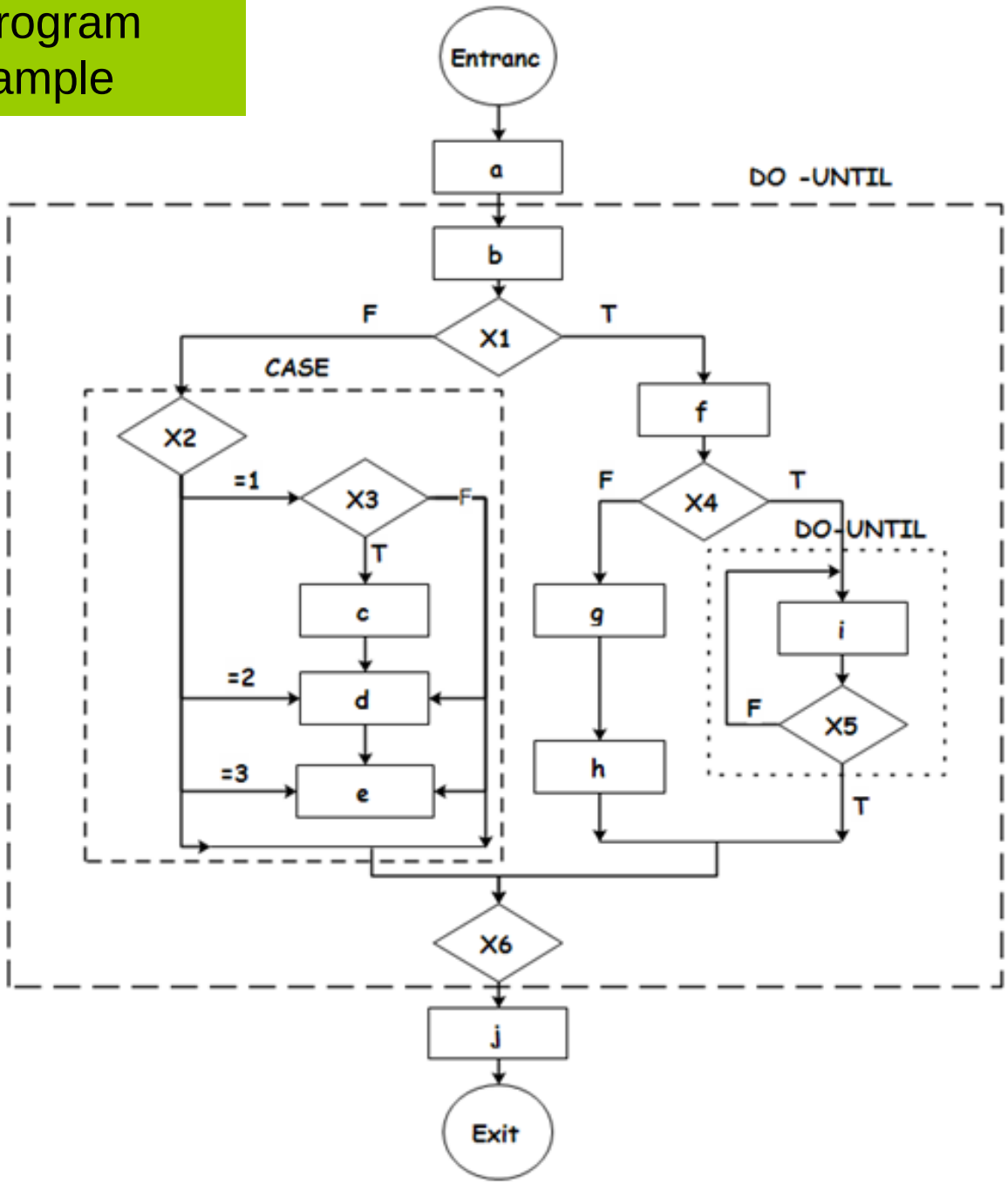


(4). DO-UNTIL

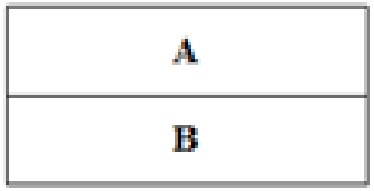


(5). CASE

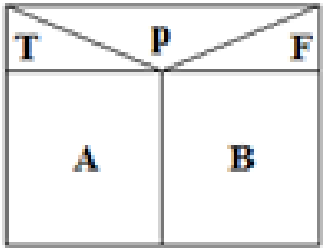
Complicated Program Flowchart Example

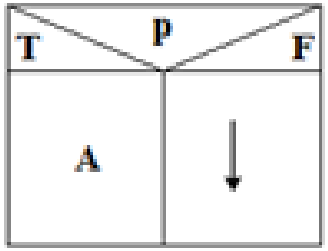


Five Basic Control Structures of N-S Diagram

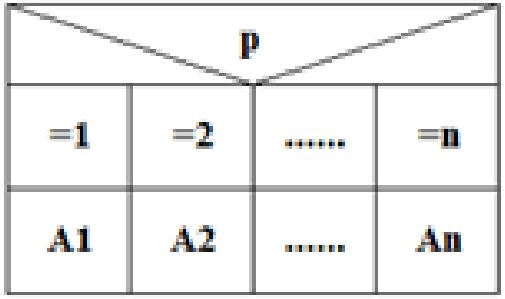


(1). SEQUENCE

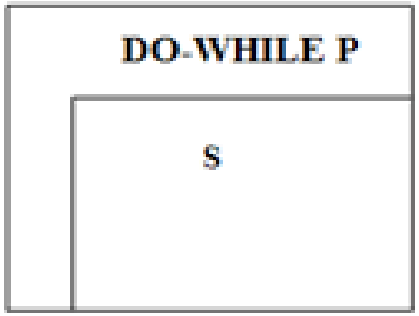




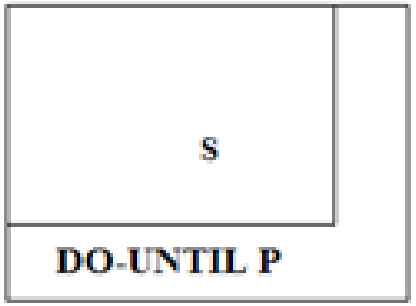
(2). SELECTION



(5). CASE

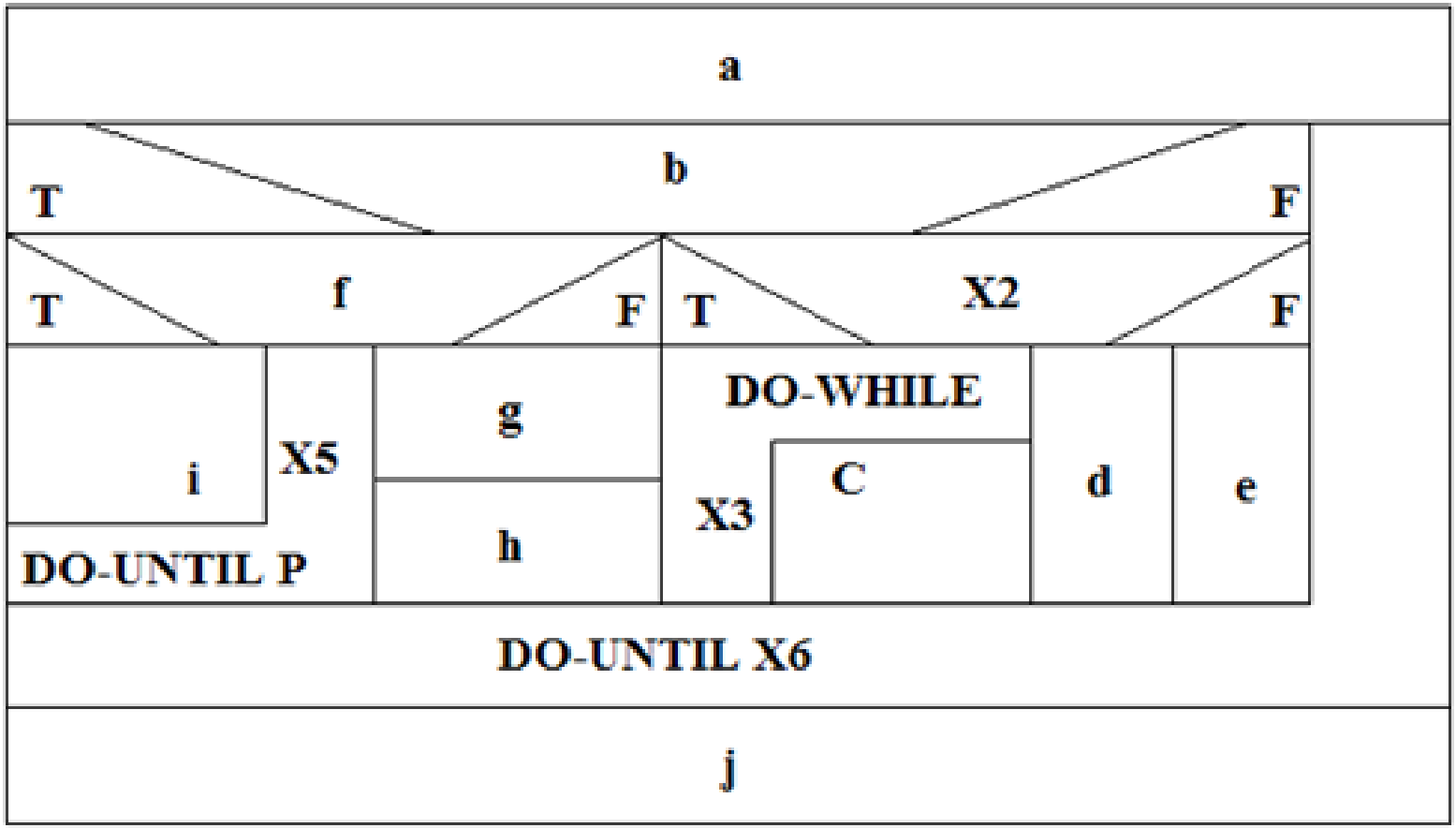


(3). WHILE

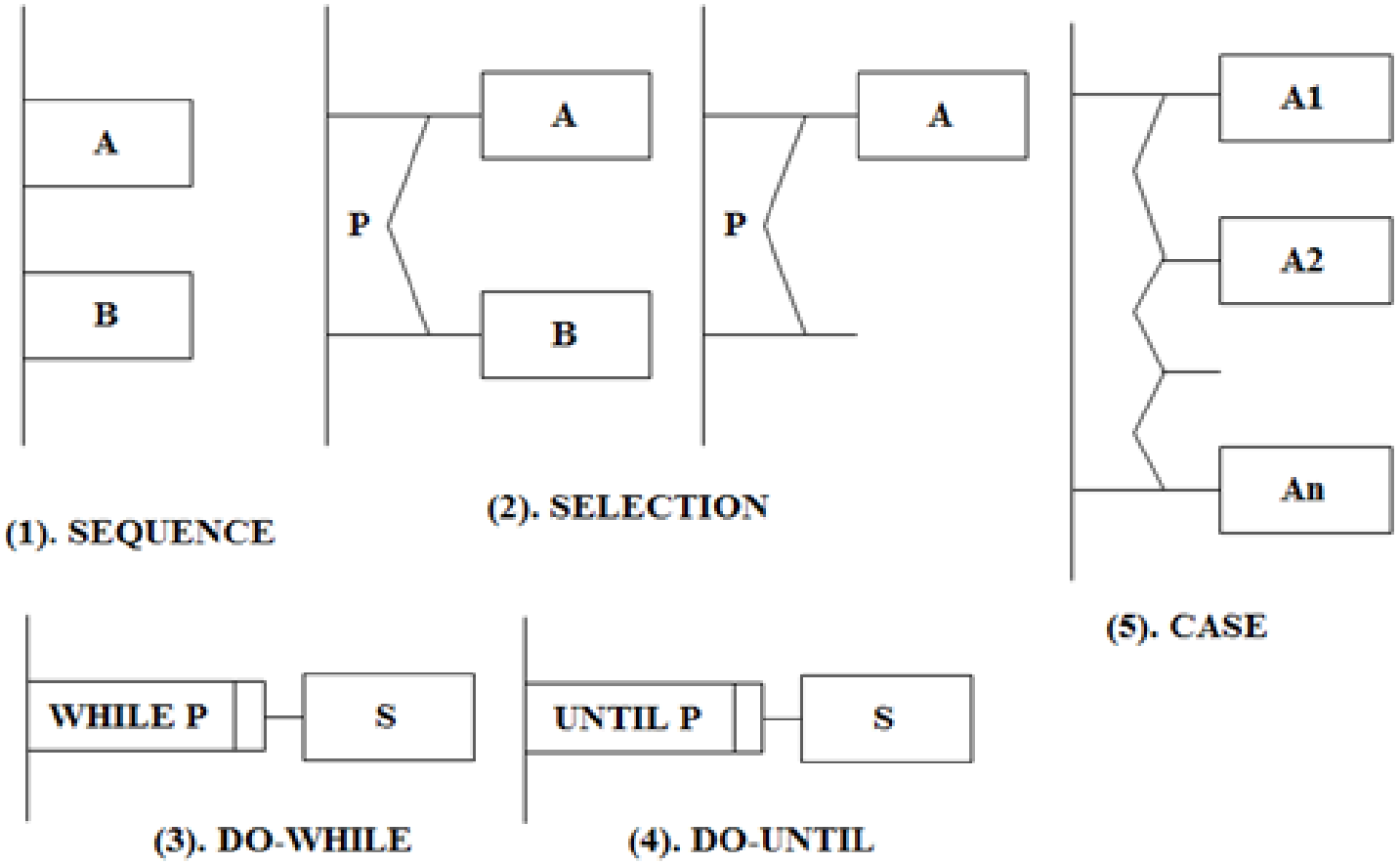


(4). UNTIL

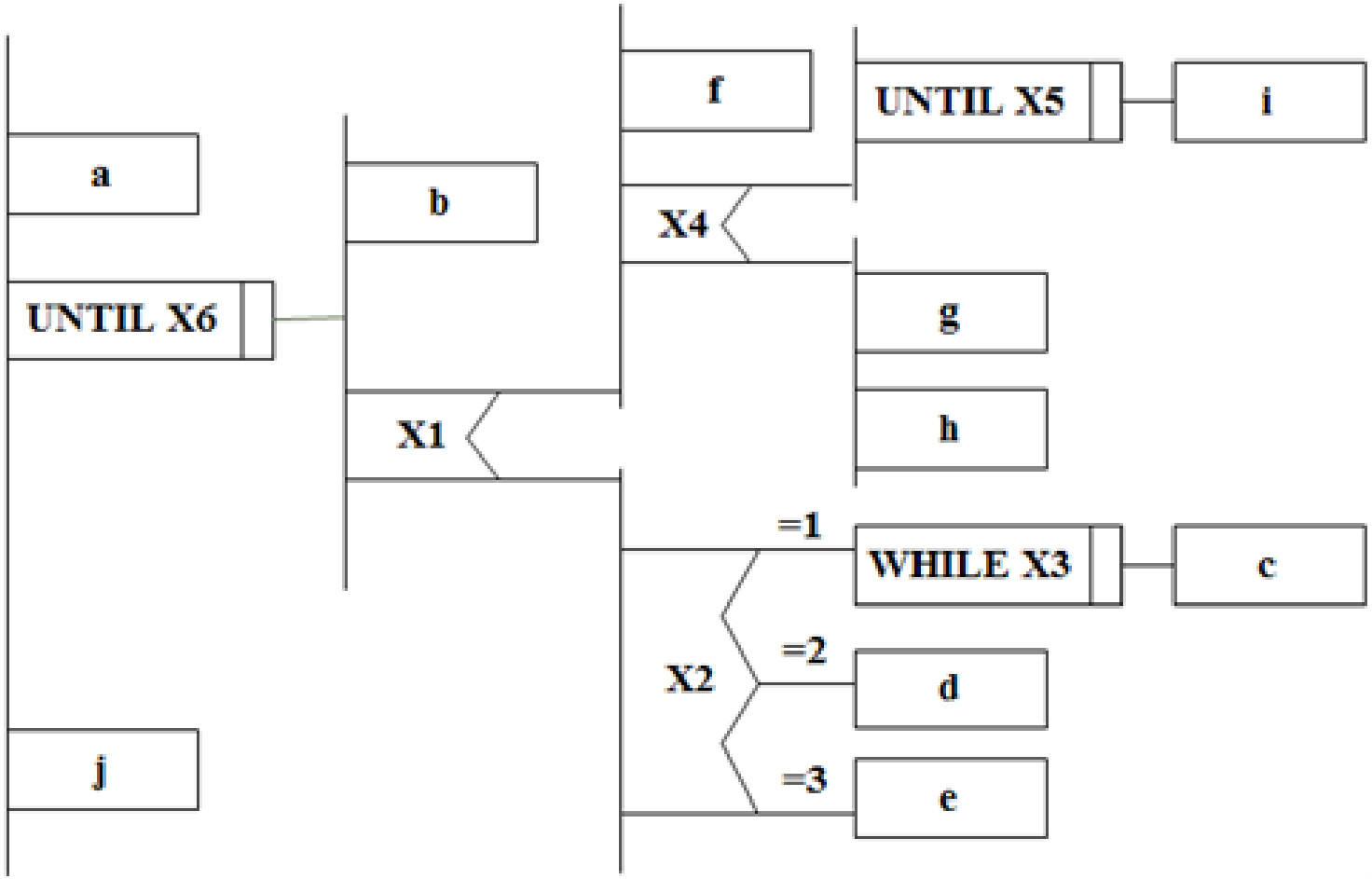
An N-S Diagram Example



Basic Control Structures of PAD



A PAD Example



Structured & OO Programming

- OO programming
  - object-oriented programming integrates code and data using the concept of an "object"
  - an object has both state (data) and behavior (code)
  - object-oriented programming attempts to provide a model for programming based on objects

Structured & OO Programming

- Objects are designed in class hierarchies
  - a class is the type of a class of objects
- Object orientation uses encapsulation and information hiding
- Polymorphism and inheritance are the main features of OO programming
- Design patterns are helpful usually in most OO program designs
  - GoF patterns, GRASP patterns
- UML diagrams can be used to express OO program design ideas

Structured & OO Programming

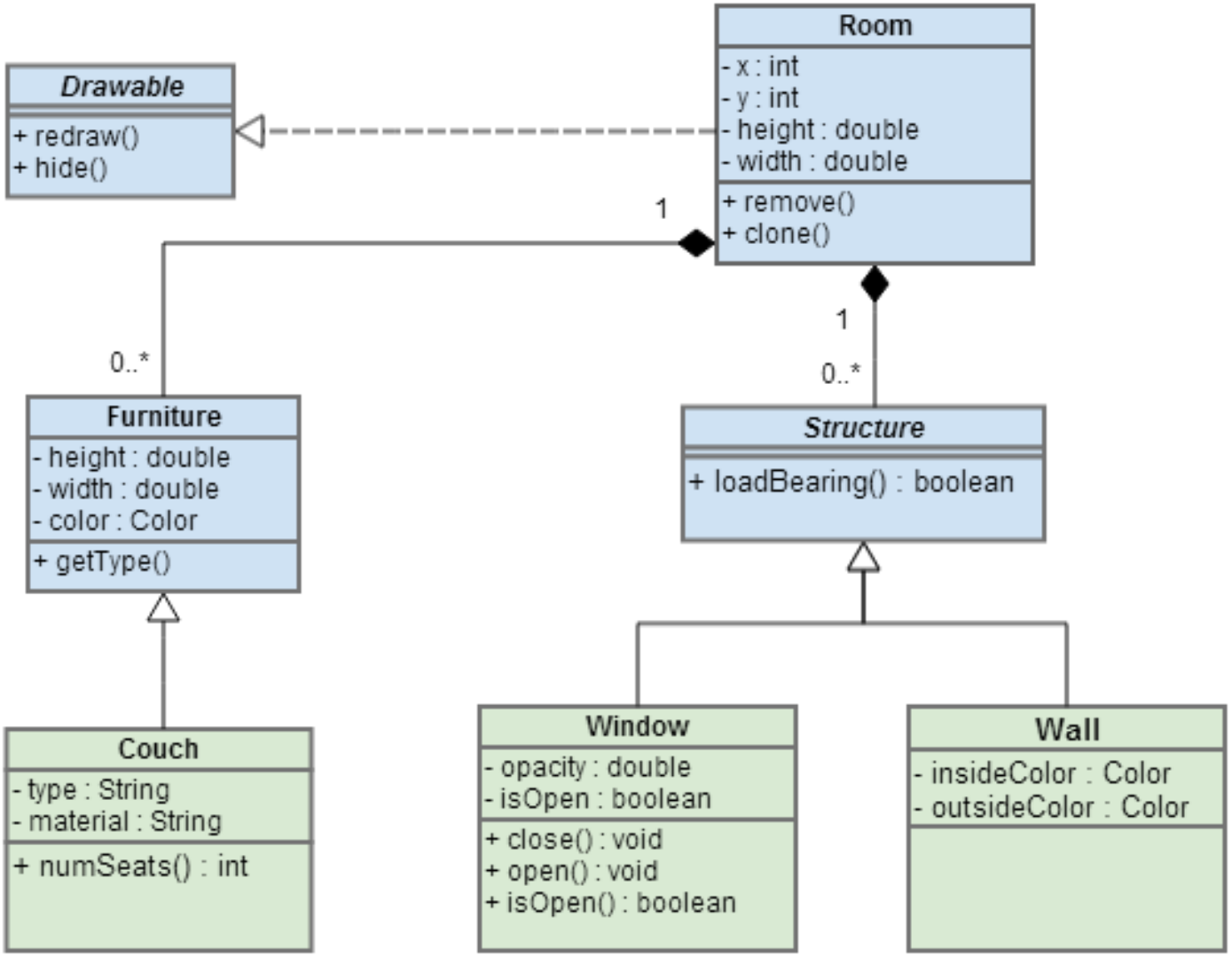
- UML class diagram
  - a type of static structure diagram that describes the structure of a system
  - by showing the system's classes, their attributes, operations (or methods), and the relationships among objects
  - classes are represented with boxes which contain three parts
    - name of the class, attributes of the class, methods or operations the class can take

Structured & OO Programming

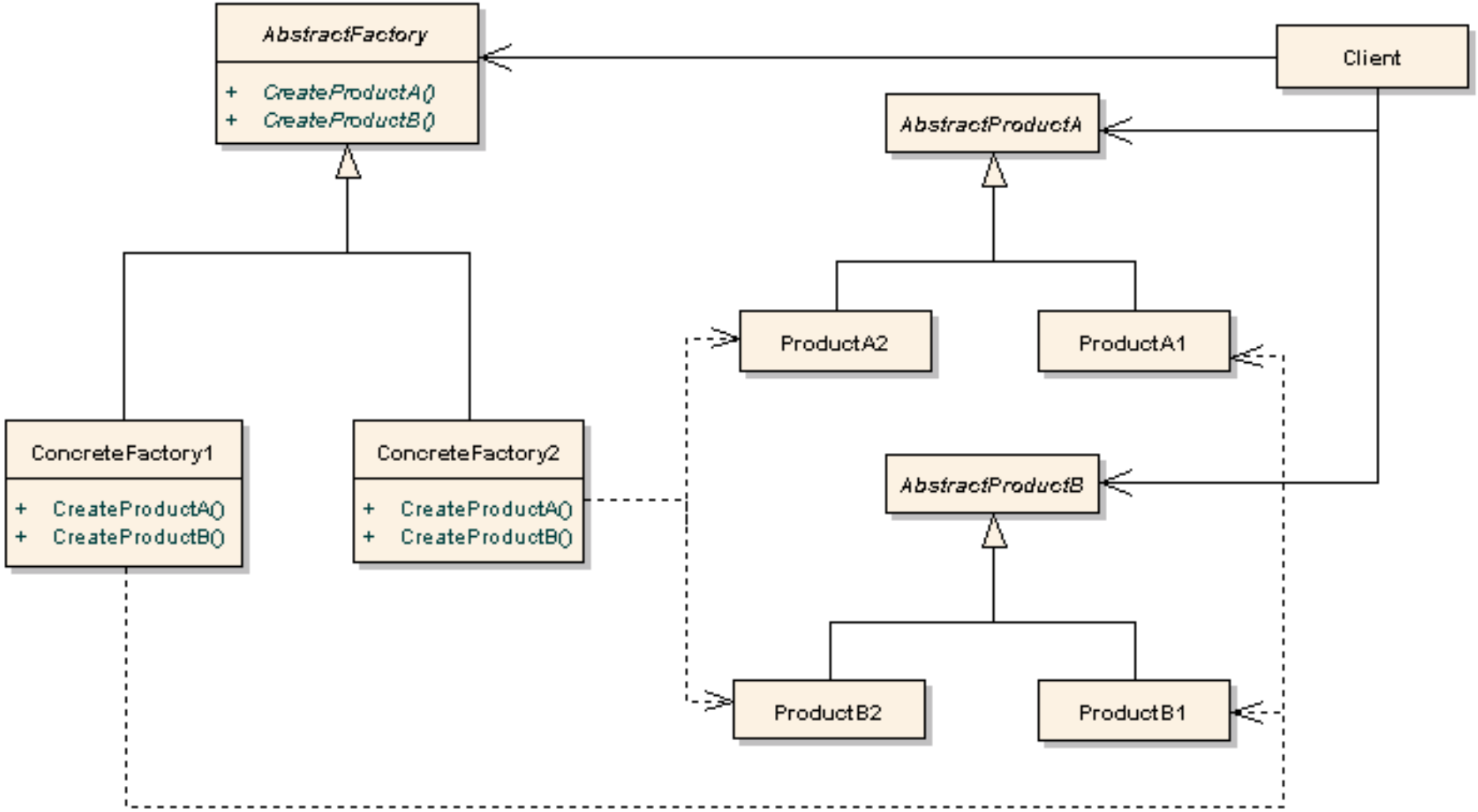
- UML class diagram (cont.)
  - Relationship
    - Instance level relationships
      - link, association, aggregation, composition
    - Class level relationships
      - gerneralization, realization, dependency
  - Visibility

+	Public
−	Private
#	Protected
/	Derived (can be combined with one of the others)
~	Package

Class Diagram Sample



AbstractFactory Pattern



Programming for Performance

- In concurrent programming, there are two basic units of execution: processes and threads
- In the Java programming language concurrent programming is mostly concerned with threads
  - However, processes are also important

Programming for Performance

- A process has a self-contained execution environment, or a complete, private set of basic run-time resources
- To facilitate communication between a set of cooperating processes, most operating systems support Inter Process Communication (IPC) resources, such as pipes and sockets

Programming for Performance

- Process (cont.)
  - When an application component starts and it does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution
  - By default, all components of the same application run in the same process and thread (called the "main" thread or UI thread)

Programming for Performance

- Process (cont.)
  - The Android system tries to maintain an application process for as long as possible, and remove old processes to reclaim memory for new or more important processes
  - There are five levels in the importance hierarchy, the first one is most important
    - Foreground process, Visible process, Service process, Background process, Empty process

Programming for Performance

- Threads are sometimes called lightweight processes, existing within a process
- Creating a new thread requires fewer resources than creating a new process
- Every process has at least one thread
- Threads share the process's resources, including memory and open files

Programming for Performance

- In Android, when an application is launched, the system creates a thread of execution for the application, called "main" , which is also sometimes called the UI thread
- Two Rules
  - Do not access the Android UI toolkit from outside the UI thread
  - Do not block the UI thread

Programming for Performance

- ANR (Application Not Responding)
  - Service timeout: 20s
  - Broadcast timeout: 10s
  - Event-dispatching timeout: 5s
  - ContentProvider timeout: 20s
- Avoiding ANR
  - Should perform long task on worker thread
  - Use handler to communicate between ui thread and work thread

Programming for Performance

- If you have operations to perform that are not instantaneous, you should do them in separate threads ("background" or "worker" threads) in Android
  - You can use **View.post(Runnable)** method to access UI compoment outside UI thread
  - Or use handler, looper, message queue to deliver messages between UI thread and worker thread

Using **View.post(Runnable) Demo**

```
@Override
public void onClick(View v) {
    if (v.getId() == R.id.start) {
        ThreadDemo td = new ThreadDemo();
        td.start();
    }
```

```
TextView tv;

class ThreadDemo extends Thread {
    ...
    @Override
    public void run() {
        while (run) {
            ...
            tv.post(new Runnable() {
                @Override
                public void run() {
                    tv.setText("new content");
                }
            });
            ...
        }
    }

}
```

Causes the Runnable to be added to the message queue. The runnable will be run on the user interface thread

Programming for Performance

- Message Queue
  - Low-level class holding the list of messages to be dispatched by a Looper
- Looper
  - Class used to run a message loop for a thread
  - Most interaction with a message loop is through the Handler class

Programming for Performance

- Handler
  - A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue
  - When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it

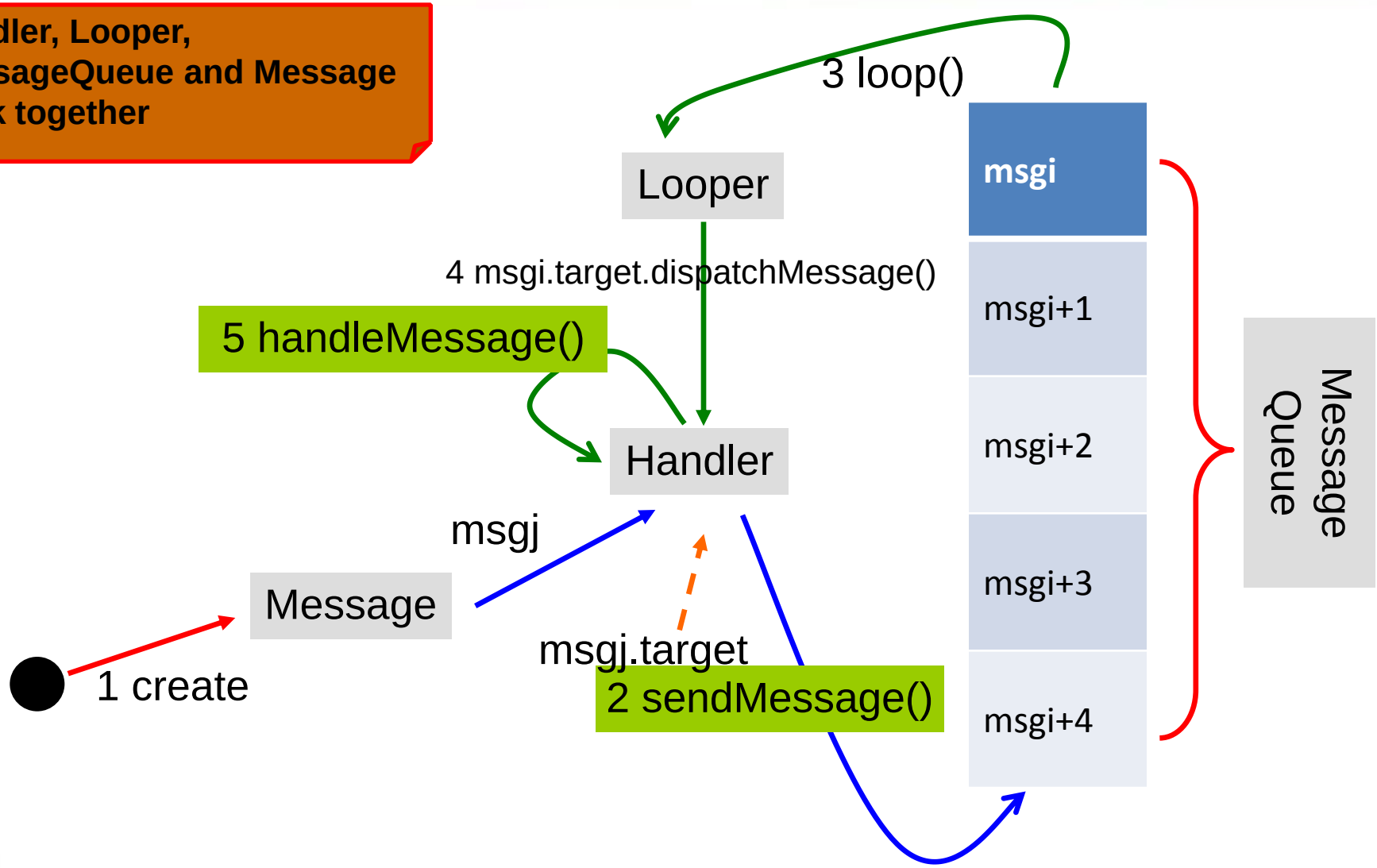
Using **Handler**

```java
TextView tv;
Handler h = new Handler() {

    @Override
    public void handleMessage(Message msg) {
        tv.setText(String.valueOf(msg.getData()
                .get("new_content")));
    }
```

```java
class ThreadDemo extends Thread {
    boolean run = true;
    @Override
    public void run() {
        while (run) {
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                run = false;
            }
            Message m = h.obtainMessage();
            Bundle data = new Bundle();
            data.putCharSequence("new_content", "new Value");
            m.setData(data);
            h.sendMessage(m);
        }

    }
}
```

```java
@Override
public void onClick(View v) {
    if (v.getId() == R.id.start) {
        ThreadDemo td = new ThreadDemo();
        td.start();
    }
}
```

Handler, Looper, MessageQueue and Message work together



Programming for Performance

- AsyncTask
  - In android, AsyncTask allows you to perform asynchronous work on your user interface
  - To use it, you must subclass AsyncTask and implement the doInBackground() callback method, which runs in a pool of background threads
  - It handles result in callback method **onPostExecute** (Result result)

Programming for Performance

- AsyncTask (cont.)
  - Rules
    - The AsyncTask class must be loaded on the UI thread
    - The task instance must be created on the UI thread
    - execute(Params...) must be invoked on the UI thread
    - Do not call onPreExecute(), onPostExecute(Result), doInBackgro und(Params...),onProgressUpdate(Progress...) manually
    - **The task can be executed only once**

```
class MyTask extends AsyncTask<Params, Progress, Result> { ... }
```

Using **AsyncTask**

```java
TextView tv;
@Override
public void onClick(View v) {
    if (v.getId() == R.id.start) {
        MyTaskAsync ta = new MyTaskAsync();
        ta.execute(10);
    }
}
```

```java
class MyTaskAsync extends AsyncTask<Integer, Integer, Integer> {
    @Override
    protected Integer doInBackground(Integer... params) {
        int total = params[0];
        for (int i = 0; i < 10; i++) {
            total = total + i;
            publishProgress(i);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        return total;
    }
    @Override
    protected void onPostExecute(Integer result) {
        tv.setText(String.valueOf(result));
    }
}
```

Programming for Performance

- IPC
  - Android offers a mechanism for interprocess communication (IPC) using remote procedure calls (RPCs)
  - And provides all the code to perform these IPC transactions
  - To perform IPC, your application must bind to a service

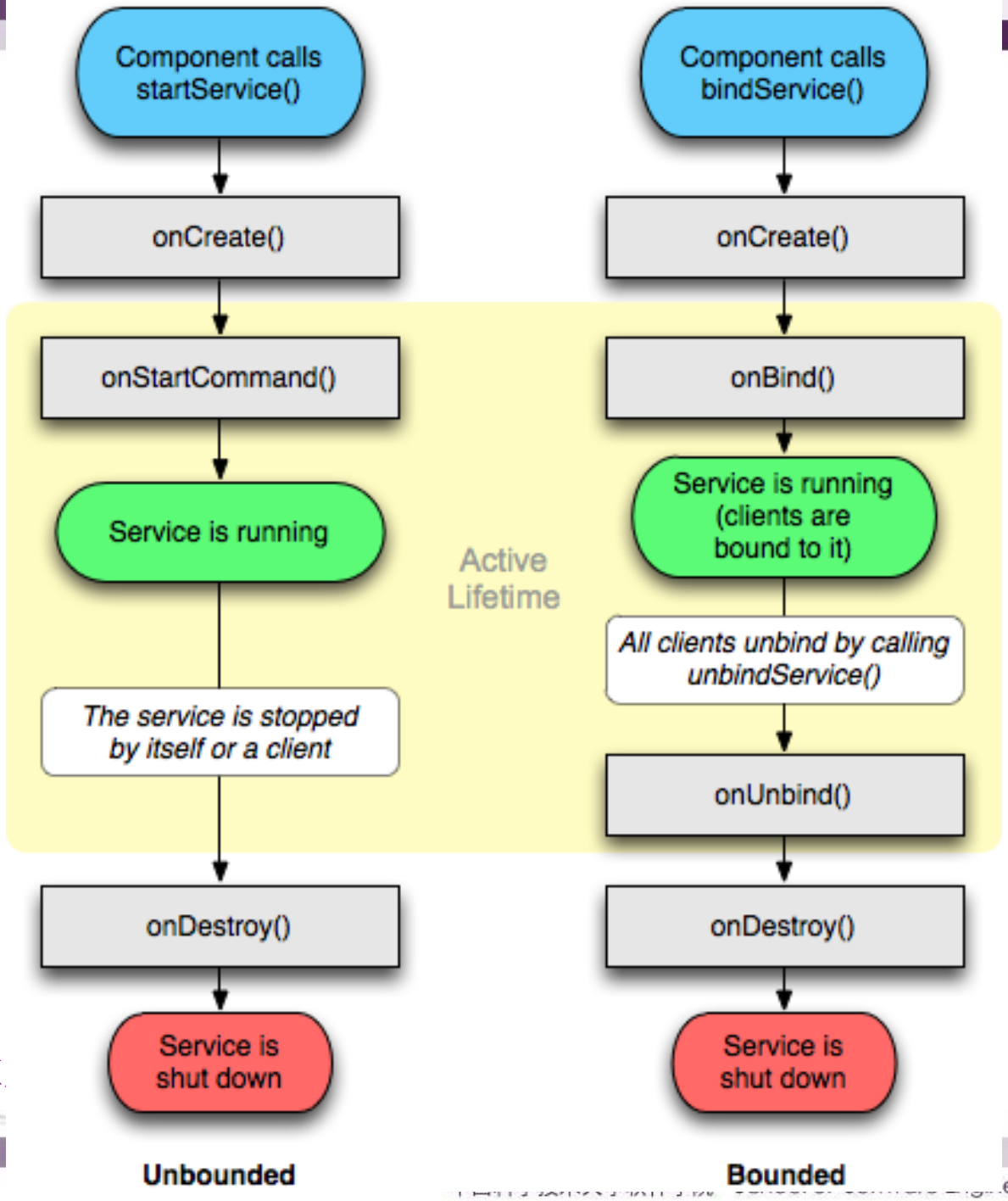
Programming for Performance

- A Service is an application component that can perform long-running operations in the background and does not provide a user interface
- A service can essentially take two forms
  - Started
  - Bound
- Android allowing you to create a new process and specify your service running in

Programming for Performance

- Started Service
  - Started by calling startService()
  - A started service performs a single operation and does not return a result to the caller
  - The service should stop itself, when the operation is done
- Bound Service
  - Bound to components by calling bindService()
  - A bound service runs only as long as another application component is bound to it
  - Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed

The Service Lifecycle



Programming for Performance

- Creating a Started Service
  - There are two classes you can extend to create a started service
    - Service
      - This is the base class for all services. It's important that you create a new thread in which to do all the service's work
    - IntentService
      - This is a subclass of Service that uses a worker thread to handle all start requests, one at a time
      - **The constructor of subclass should have no parameters and call constructor of super class**

```java
Intent intent = new Intent(ServiceTestingActivity.this, ServiceDemo.class);
startService(intent);
```

Extending the Service Class

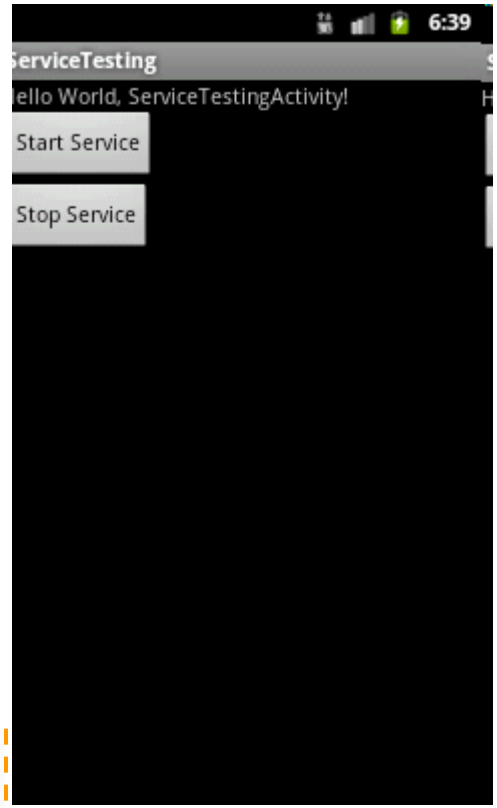
```java
public class ServiceDemo extends Service {
    private MessageHandler mh;
    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public void onCreate() {
        HandlerThread ht = new HandlerThread("ServiceStartArguments",
                android.os.Process.THREAD_PRIORITY_BACKGROUND);
        ht.start();
        mh = new MessageHandler(ht.getLooper());
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Message msg=mh.obtainMessage();
        msg.arg1=startId;
        mh.sendMessage(msg);
        return Service.START_STICKY;
    }
    private final class MessageHandler extends Handler {
        public MessageHandler(Looper loop) {
            super(loop);
        }
        @Override
        public void handleMessage(Message msg) {
            long endTime = System.currentTimeMillis() + 5 * 1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(1000);
                        Log.v("running", "Service is running!");
                    } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }
            stopSelf(msg.arg1);
        }
    }}
```

int	START_CONTINUATION_MASK
int	START_FLAG_REDELIVERY
int	START_FLAG_RETRY
int	START_NOT_STICKY
int	START_REDELIVER_INTENT
int	START_STICKY
int	START_STICKY_COMPATIBILITY

Extending the IntentService Class

```java
Intent intent = new Intent(ServiceTestingActivity.this, IntentServiceDemo.class);
startService(intent);
```

```java
public class IntentServiceDemo extends IntentService {

    public IntentServiceDemo() {
        super("TestIntentService");
    }
    @Override
    protected void onHandleIntent(Intent intent) {
        long endTime = System.currentTimeMillis() + 5 * 1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(1000);
                    Log.v("running", "Service is running!");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```




```manifest
<manifest ... >
  ...
  <application ... >
    <service
android:name="water.java.service.IntentServiceDemo"></service>
    ...
  </application>
</manifest>
```

Programming for Performance

- Creating a Bound Service
  - A bound service is the server in a client-server interface
  - A client can bind to the service by calling bindService()
    - When it does, it must provide an implementation of ServiceConnection, which monitors the connection with the service
  - Multiple clients can connect to the service at once
    - When the last client unbinds from the service, the system destroys the service
  - Extending the Binder Class
    - If your service is private to your own application and runs in the same process as the client (Local Service)
  - Using a Messenger
    - If you need your interface to work across different processes (Remote Service)

Extending the Binder Class

```java
@Override
protected void onStart() {
    // TODO Auto-generated method stub
    super.onStart();
    Intent intent = new Intent(this,
            LocalService.class);
    bindService(intent, sc, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    if (isBound) {
        unbindService(sc);
        isBound = false;
    }
}
```

```java
public class LocalService extends Service {
    private IBinder ib=new LocalBinder();;

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return ib;
    }
    public class LocalBinder extends Binder {
        public LocalService getService() {
            return LocalService.this;
        }
    }

    public int getThreadId() {
        return android.os.Process.myTid();
    }
}
```

```java
private boolean isBound=false;
private LocalService ls;
private ServiceConnection sc = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        LocalBinder lb = (LocalBinder) service;
        ls = lb.getService();
        isBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        isBound = false;
    }
};
```

u.cn/~waterzhj

Using Messenger

```java
@Override
protected void onStart() {
    // TODO Auto-generated method stub
    super.onStart();
    Intent intent = new Intent(this,
            MessengerService.class);
    bindService(intent, sc, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    if (isBound) {
        unbindService(sc);
        isBound = false;
    }
}
```

```java
public class MessengerService extends Service {

    private final Messenger msger = new Messenger(new MessageHandler());

    @Override
    public IBinder onBind(Intent arg0) {
        return msger.getBinder();
    }

    class MessageHandler extends Handler {

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
            case 0: {
                Toast.makeText(getApplicationContext(), "this is 0", 2000)
                        .show();
                break;
            }
            case 1: {
                Toast.makeText(getApplicationContext(), "this is 1", 2000)
                        .show();
                break;
            }
            default: {
                super.handleMessage(msg);
            }
            }
        }
    };
}
```

```java
private boolean isBound = false;
private Messenger msger;
private ServiceConnection sc = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        msger = new Messenger(service);
        isBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        msger = null;
        isBound = false;
    }
};
```

```java
if (isBound) {
    Message msg = Message.obtain(null, 0, 0, 0);
    try {
        msger.send(msg);
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Programming for Performance

- Broadcast Receivers
  - A broadcast receiver is a component that responds to system-wide broadcast announcements
  - A broadcast receiver is implemented as a subclass of BroadcastReceiver
  - There are two major classes of broadcasts that can be received
    - **Normal broadcasts**
    - **Ordered broadcasts**

Programming for Performance

- Receiver Lifecycle
  - A BroadcastReceiver object is only valid for the duration of the call to onReceive(Context, Intent)
- Permissions
  - Access permissions can be enforced by either the sender or receiver of an Intent
    - when sending, you supply a non-null permission to sendBroadcast(Intent intent, String permission)
    - when receiving, you supply a non-null *permission* to registerReceiver(BroadcastReceiver br, IntentFilter filter, String permission, android.os.Handler handler)

Using BroadcastReceiver

```java
Intent intent = new Intent("water.java.broadcastreceiver");
intent.putExtra("broadcast", "Test broadcast");
sendBroadcast(intent);
```

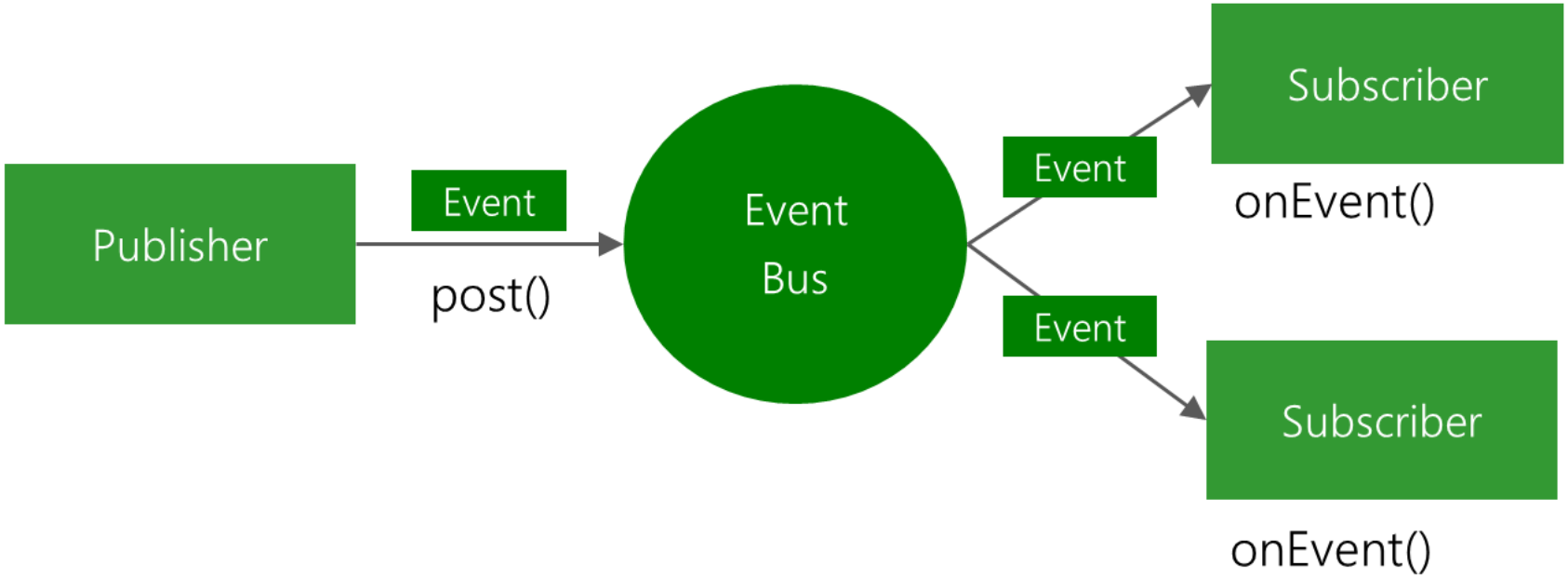
```java
public class BroadcastReceiverDemo extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (action.equals("water.java.broadcastreceiver")) {
            Toast.makeText(context, intent.getStringExtra("broadcast"), 2000)
                    .show();
        } else if (action.equals("android.provider.Telephony.SMS_RECEIVED")) {
            Toast.makeText(context, "new message!", 2000).show();
        }
    }

}
```

```androidmanifest
<uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
<receiver android:name="water.java.service.BroadcastReceiverDemo">
    <intent-filter>
        <action android:name="water.java.broadcastreceiver" />
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

Programming for Performance

- EventBus
  - open source (by greenrobot)
  - simplifying the communication between components
    - decouples event senders and receivers
    - performs well with Activities, Fragments, and background threads
    - avoids complex and error-prone dependencies and life cycle issues
  - substitute for Intent, Handler/Message Queque, Broadcast etc.

EventBus architecture



Programming for Performance

- EventBus (cont.)
  - defining event message
    - POJO without any requirements
  - preparing subscribers
    - using @Subscribe to annotate the message handling method
    - registering subscribers
    - unregistering subscribers
  - post message
    - using EventBus.getDefault().post() api

Using EventBus

```java
public class EventMessage {
    private String content;
    public void setContent(String content) { this.content = content; }
    public String getContent() { return content; }
}
```

Message

```java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {...}
    @Override
    protected void onStart() {
        EventBus.getDefault().register(this);
        super.onStart();
    }
    @Override
    protected void onStop() {
        EventBus.getDefault().unregister(this);
        super.onStop();
    }
    @Subscribe(threadMode=ThreadMode.MAIN)
    public void onReceiveEvent(EventMessage me){
        //handling me
    } }
```

Receiver/Subscriber

Sender/Publisher

```java
public void send(View v){
    EventMessage event=new EventMessage();
    event.setContent(messageContent);
    EventBus.getDefault().post(event);
}
```

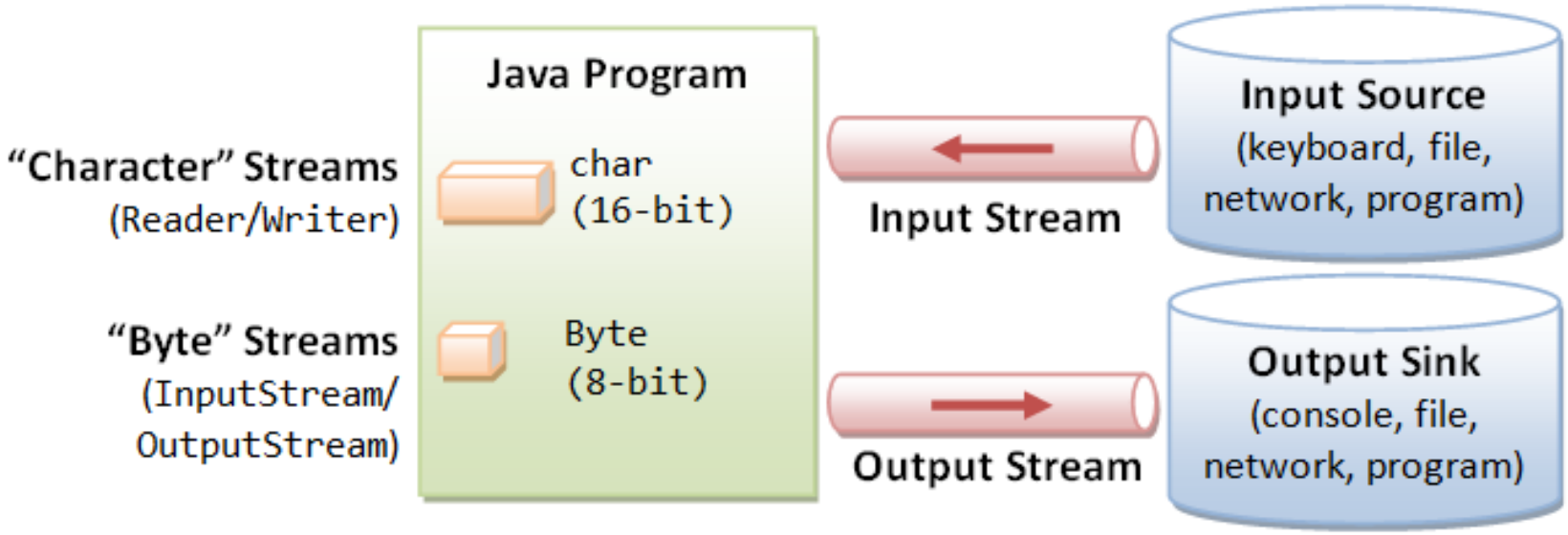
File Programming

- Stream I/O
  - Programs read inputs from data sources (e.g., keyboard, file, network, memory buffer, or another program) and write outputs to data sinks
  - In Java standard I/O, inputs and outputs are handled by the so-called streams
  - A stream is a sequential and contiguous one-way flow of data

File Programming

- Stream I/O operations involve three steps
  - Open an input/output stream associated with a physical device by constructing an appropriate I/O stream instance
  - Read from the opened input stream until "end-of-stream" encountered, or write to the opened output stream
  - Close the input/output stream

Two Type Streams In Java



Internal Data Formats:

- Text (char): UCS-2
- int, float, double, etc.

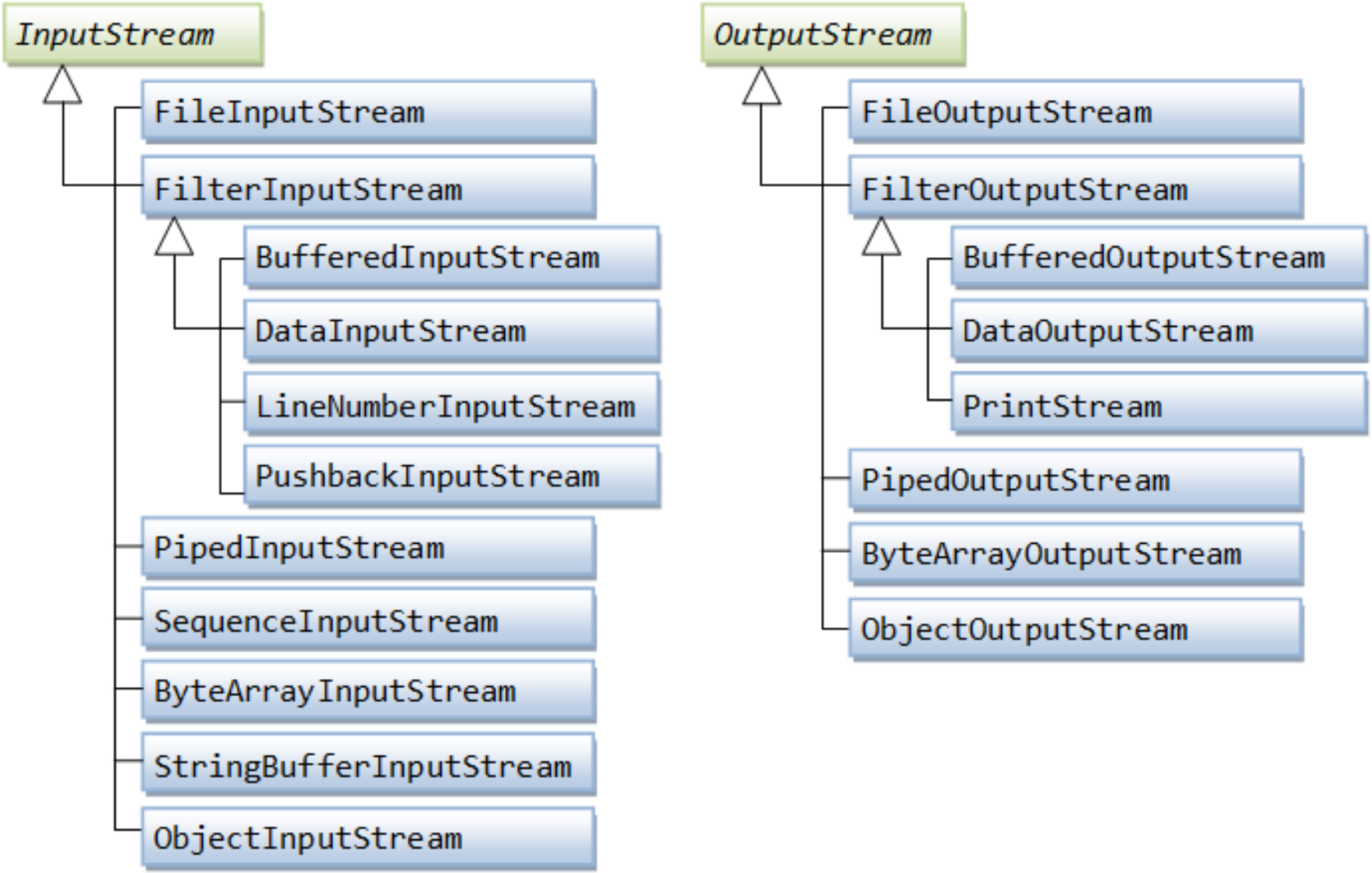
External Data Formats:

- Text in various encodings (US-ASCII, ISO-8859-1, UCS-2, UTF-8, UTF-16, UTF-16BE, UTF16-LE, etc.)
- Binary (raw bytes)

File Programming

- Byte-Based I/O & Byte Streams
  - Byte streams are used to read/write raw bytes serially from/to an external device
  - All the byte streams are derived from the abstract superclasses InputStream and OutputStream
  - InputStream declares an abstract method read() to read one data-byte from the input source
  - OutputStream declares an abstract method write() to write a data-byte to the output sink

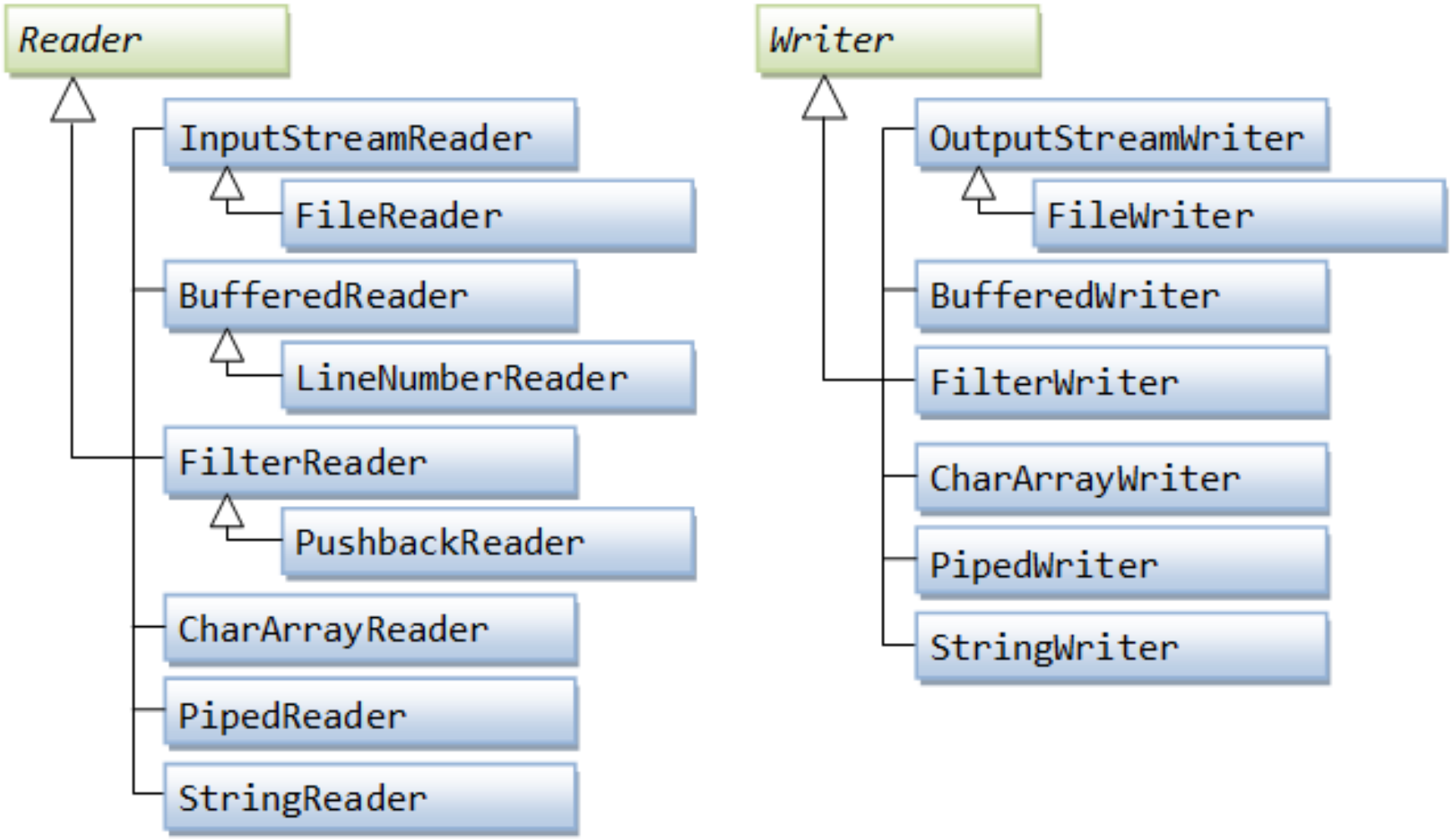
Class Hiararhies of Byte Streams



File Programming

- Character-Based I/O & Character Streams
  - Java internally stores characters (char type) in 16-bit UCS-2 character set
  - Java has to differentiate between byte-based I/O for processing 8-bit raw bytes, and character-based I/O for processing texts
  - The character streams needs to translate between the character set used by external I/O devices and Java internal UCS-2 format
  - Instead of InputStream and OutputStream, we use Reader and Writer for character-based I/O

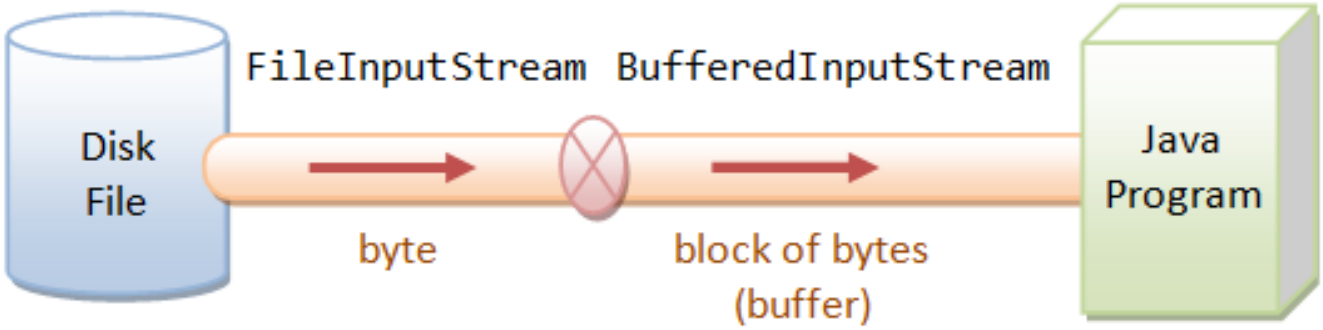
Class Hiararhies of Character Streams

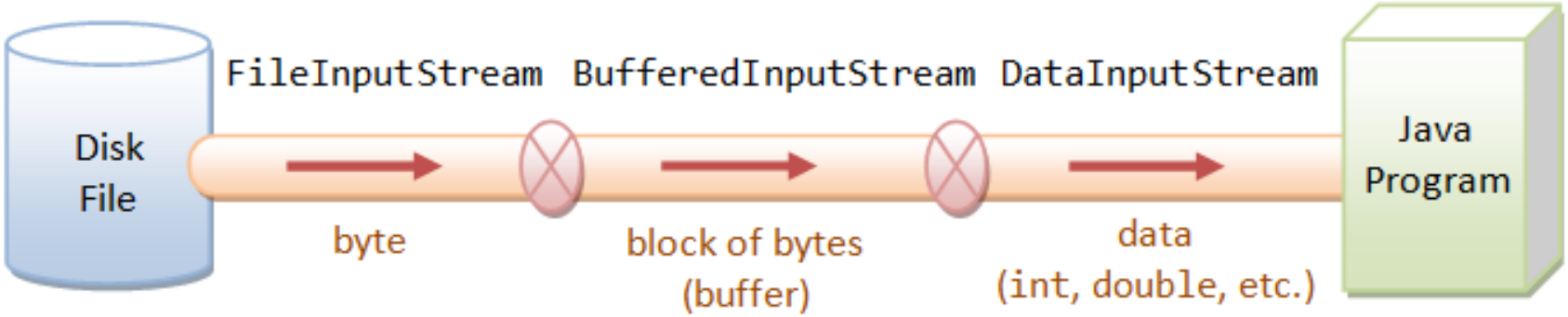


File Programming

- File I/O
  - FileInputStream and FileOutputStream are concrete implementations to the abstract classes InputStream and OutputStream, to support I/O from disk files
  - FileReader and FileWriter are concrete implementations to the abstract superclasses Reader and Writer, to support I/O from disk files

Layered (or Chained) File I/O Streams





File Programming

- File and Directory
  - Java File class can represent either a file or a directory
  - A path string is used to locate a file or a directory, which is system dependent
    - for example, Windows use back-slash '\' as the directory separator; while Unixes/Mac use forward-slash '/'
  - A path could be absolute (beginning from the root) or relative

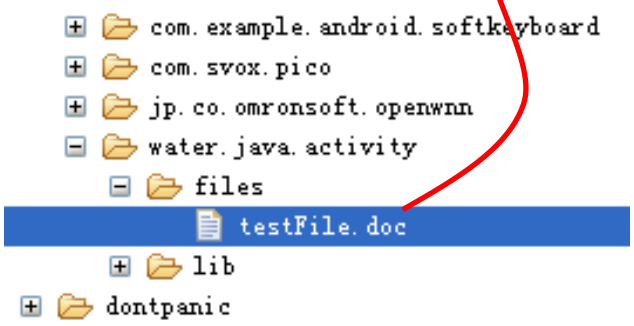
File Programming

- Android file programming
  - You can save files directly on the device's internal storage
  - By default, files saved to the internal storage are private to your application and other applications cannot access them
  - When the user uninstalls your application, these files are removed

File Programming

- Create/Write/Read/Delete
  - To create and write a private file to the internal storage
    - Call openFileOutput() with the name of the file and the operating mode
    - Write to the file with write()
    - Close the stream with close()
  - To read a file from internal storage
    - Call openFileInput() and pass it the name of the file to read
    - Read bytes from the file with read()
    - Then close the stream with close()
  - To delete a file from internal storage
    - Call File.delete()

Using File Storage




```java
String fileName = "testFile.doc";
String fileContent = "testFileContent";

public void createAndWriteFile() {
    try {
        FileOutputStream fos = openFileOutput(fileName,
                Context.MODE_PRIVATE);
        fos.write(fileContent.getBytes());
        fos.flush();
        fos.close();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void readFile() {
    try {
        FileInputStream fis = openFileInput(fileName);
        byte[] buffer = new byte[1024];
        fis.read(buffer);
        fis.close();
        tv.setText(new String(buffer));
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public boolean deleteFile() {
    String filepath = getFilesDir().getAbsolutePath();
    File file = new File(filepath + "//" + fileName);
    boolean result = file.delete();
    return result;
}
```

File Programming

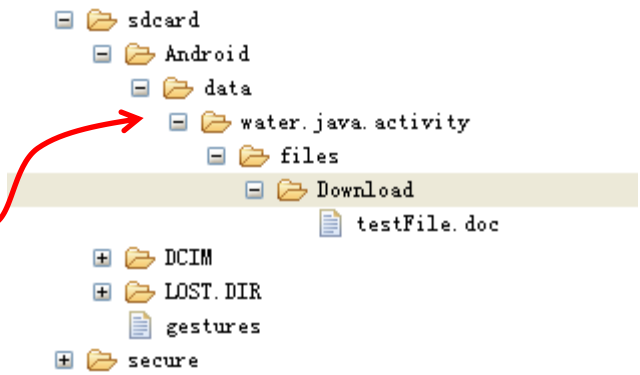
- Using the External Storage
  - Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer
  - Checking media availability
    - Before you do any work with the external storage, you should always call getExternalStorageState() to check whether the media is available

File Programming

- Using the External Storage (cont.)
  - Accessing files on external storage
    - If you're using API Level 8 or greater, use getExternalFilesDir() to open a File that represents the external storage directory where you should save your files
    - If the user uninstalls your application, this directory and all its contents will be deleted

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
```

Creating Files on External Storage




```java
String fileName = "testFile.doc";
String fileContent = "testFileContent";

public void createFileOnSDCard() {
    String state = Environment.getExternalStorageState();
    if (state.equals(Environment.MEDIA_MOUNTED)) {
        File externalFile = getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS);
        File newFile = new File(externalFile.getAbsoluteFile() + "//"
                + fileName);
        try {
            boolean result = newFile.createNewFile();
            if (result) {
                FileOutputStream fos = new FileOutputStream(newFile);
                fos.write(fileContent.getBytes());
                fos.flush();
                fos.close();
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

aterzhj

File Programming

- SharedPreferences
  - The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types
    - Modifications to the preferences must go through an SharedPreferences.Editor object to ensure the preference values remain in a consistent state and control when they are committed to storage
    - use SharedPreferences methods such as getBoolean() and getString()
      - Map, Set, boolean, int, long, float, String

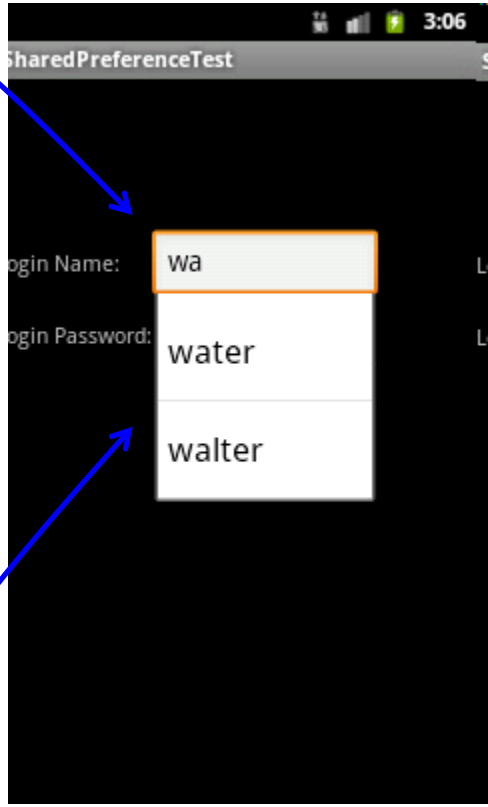
File Programming

- SharedPreferences (cont.)
  - To get a SharedPreferences object for your application, use one of two methods
    - getSharedPreferences()
      - Use this if you need multiple preferences files identified by name, which you specify with the first parameter
    - getPreferences()
      - Use this if you need only one preferences file for your Activity
  - To write values
    - Call edit() to get a SharedPreferences.Editor
    - Add values with methods such as putBoolean() and putString()
    - Commit the new values with commit()

Using SharedPreferences

```
act = (AutoCompleteTextView) findViewById(R.id.loginName);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line, initAdapter());
act.setAdapter(adapter);
```

```
@Override
protected void onStop() {
    super.onStop();
    SharedPreferences sp = getPreferences(Activity.MODE_PRIVATE);
    String text = act.getText().toString();
    if (!(list.contains(text) || text.equals("") || text == null)) {
        SharedPreferences.Editor edit = sp.edit();
        edit.putString("name" + list.size(), text);
        edit.commit();
    }
}

ArrayList<String> list;

public String[] initAdapter() {
    String[] result = null;
    list = new ArrayList<String>();
    SharedPreferences sp = getPreferences(Activity.MODE_PRIVATE);
    int i = 0;
    boolean exists = sp.contains("name" + i);
    while (exists) {
        list.add(sp.getString("name" + i, null));
        i++;
        exists = sp.contains("name" + i);
    }
    int size = list.size();
    result = new String[size];
    int j = 0;
    for (String item : list) {
        result[j] = item;
        j++;
    }
    return result;
}
```




```
<AutoCompleteTextView android:layout_width="150dp"
    android:layout_height="wrap_content" android:id="@+id/loginName" />
```

tc.edu.cn/ waterznj

SharedPreferences File

com.example.android.livecubes		2011-11-10	02:47	drwxr-x--x
com.example.android.softkeyboard		2011-11-10	02:47	drwxr-x--x
com.svox.pico		2011-09-14	02:12	drwxr-x--x
jp.co.omronsoft.openwnn		2011-11-10	02:52	drwxr-x--x
water.java.activity		2011-11-10	03:05	drwxr-x--x
databases		2011-11-01	03:33	drwxrwx--x
lib		2011-11-10	03:05	drwxr-xr-x
shared_prefs		2011-11-10	03:06	drwxrwx--x
SharedPreferenceTestActivity.xml	245	2011-11-10	03:06	-rw-rw----
dontpanic		2011-09-14	02:11	drwxr-x---
local		2011-09-14	02:11	drwxrwx--x
lost+found		2011-09-14	02:11	drwxrwx---

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <map>
    <string name="name3">water</string>
    <string name="name4">walter</string>
    <string name="name1">rrrrrr</string>
    <string name="name2" />
    <string name="name0">eee</string>
  </map>
```

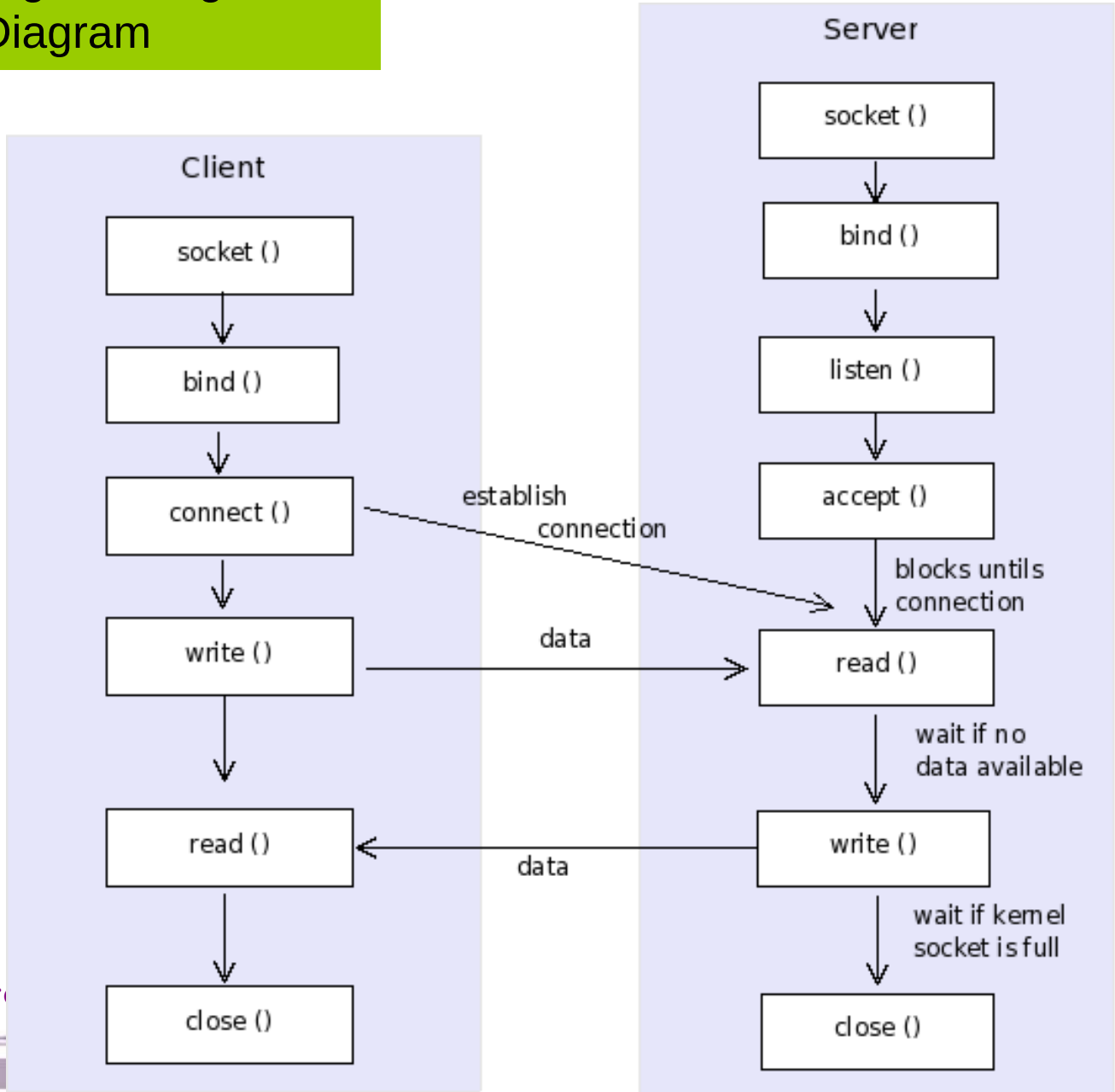
Network Programming

- The Internet is all about connecting machines together
- Sockets provide the communication mechanism between two computers
  - The socket is primarily a concept used in the Transport Layer of the Internet model
  - Java's socket model is derived from BSD (UNIX) sockets

Network Programming

- Java's socket model
  - Local socket address=IP+port
  - Remote socket address=IP+port
  - Protocol (TCP/UDP)
    - TCP: for reliable communication between two applications, used in client/server model
    - UDP: a connection-less protocol that allows for packets of data to be transmitted between applications

Socket Programming Flow Diagram



Network Programming

- Android provides access to networking in several ways, including mobile *Internet Protocol (*IP*)*, Wi-Fi, and Bluetooth
- We'll concentrate on getting your Android applications to communicate using IP network data
- Android provides a portion of the java.net package and the org.apache.httpclient package to support basic networking (deprecated)

Network Programming

- Checking the network status
  - ConnectivityManager class
    - Being used to answer queries about the state of network connectivity. It also notifies applications when network connectivity changes
    - Get an instance of this class by calling Context.getSystemService(Context.CONNECTIVITY_SERVICE)

Network Programming

- Communicating with socket
  - You need a server socket and client
    - A server socket is a stream that you can read or write raw bytes to, at a specified IP address and port
    - The client is your mobile phone based on android
  - ServerSocket class
    - represents a server-side socket that waits for incoming client connections
    - handles the requests and sends back an appropriate reply
  - Socket class
    - Provides a client-side TCP socket

Checking the network status

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE">
</uses-permission>
```

```
public boolean testConnectivity() {
    boolean result = false;
    ConnectivityManager cmg = (ConnectivityManager)
                                getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo ni = cmg.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
    if (ni.isConnected()) {
        Log.v("connected: ", ni.toString());
        result = true;
    }
    return result;
}
```

Communicating with socket

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Server

```
byte[] buffer = new byte[100];
ServerSocket server = new ServerSocket(serverPort);
Socket client = server.accept();
InputStream is = client.getInputStream();
int data = is.read(buffer);
if (data > 0) {
    System.out.println(new String(buffer));
}
is.close();
client.close();
server.close();
```

Client

```
public void connectToServer() {
    try {
        Socket server = new Socket(serverIp, serverPort);
        SocketAddress ia = server.getLocalSocketAddress();
        OutputStream os = server.getOutputStream();
        os.write(ia.toString().getBytes());
        os.flush();
        os.close();
        server.close();
    } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

}
```

Network Programming

- Working with HTTP
  - HttpURLConnection
    - An URLConnection for HTTP used to send and receive data over the web
    - Data may be of any type and length.
    - This class may be used to send and receive streaming data whose length is not known in advance
    - Obtaining a new HttpURLConnection by calling URL.openConnection()
    - Once the response body has been read, the HttpURLConnection should be closed by calling disconnect()

Using HttpURLConnection to access web service

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

```
public class ResponseAndroidClient extends HttpServlet {
```

```
public String connectToServer() {
    URL server = null;
    byte[] result = new byte[20];
    try {
        server = new URL(serverURL + "?name=" + name + "&password="
                + password);
        HttpURLConnection hc = (HttpURLConnection) server.openConnection();
        if (hc.getResponseCode() == HttpStatus.SC_OK) {
            InputStream is = hc.getInputStream();
            is.read(result);
            is.close();
        }
        hc.disconnect();
    } catch (MalformedURLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return new String(result);
}
```

Network Programming

- Working with HTTP (cont.)
  - HttpClient (Removed from 6.0)
    - encapsulates a smorgasbord of objects required to execute HTTP requests while handling cookies, authentication, connection management, and other features
    - thread safety of HTTP clients depends on the implementation and configuration of the specific client

Network Programming

- Working with HTTP (cont.)
  - HttpClient (cont.)
    - HttpGet
      - means retrieve whatever information (in the form of an entity) is identified by the Request-URI
    - HttpPost
      - is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line
    - HttpEntity
      - An entity that can be sent or received with an HTTP message. Entities can be found in some requests and in responses, where they are optional

Using HttpClient to access web service

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

```
public class ResponseAndroidClient extends HttpServlet {
```

```
public String usingHttpClient() {
    byte[] result = new byte[20];
    HttpParams hp = new BasicHttpParams();
    HttpConnectionParams.setSoTimeout(hp, 2000);
    DefaultHttpClient dhc = new DefaultHttpClient(hp);
    HttpResponse hr = null;
    HttpGet hg = new HttpGet();

    try {
        URI uri = new URI(serverURL + "?name=" + name + "&password="
                + password);
        hg.setURI(uri);
        hr = dhc.execute(hg);
        if (hr.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            HttpEntity he = hr.getEntity();
            InputStream is = he.getContent();
            is.read(result);
            is.close();
        }
    } catch (URISyntaxException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return new String(result);
}
```

Multi-media Programming

- The Android framework APIs provides a set 2D drawing APIs that allow you to render your own custom graphics
- Two ways of drawing 2D graphics
  - Draw your graphics or animations into a View object from your layout
  - Draw your graphics directly to a Canvas

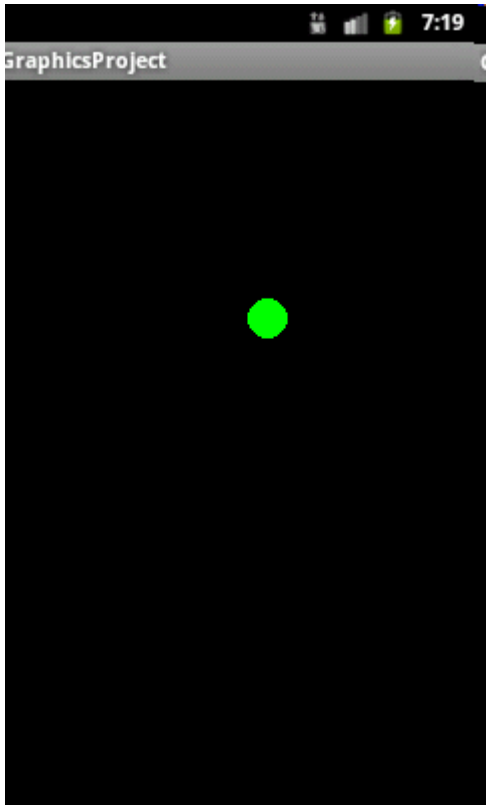
Multi-media Programming

- Draw with a Canvas
  - Via the Canvas, your drawing is actually performed upon an underlying Bitmap, which is placed into the window
  - The Canvas class has its own set of drawing methods that you can use, like drawBitmap(...), drawRect(...), and many more
  - Drawable objects that you want to put on the Canvas has its own draw() method that takes your Canvas as an argument

Drawing graphics with a canvas

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mcv = new MyCustomView(this);
    setContentView(mcv);
}
```

```java
public class MyCustomView extends View {

    private float x = 50, y = 50, r = 10;
    private Paint p = null;

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public MyCustomView(Context context) {
        super(context);
        p = new Paint();
        p.setColor(0xff00ff00);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        canvas.drawCircle(x, y, r, p);
    }
}
```



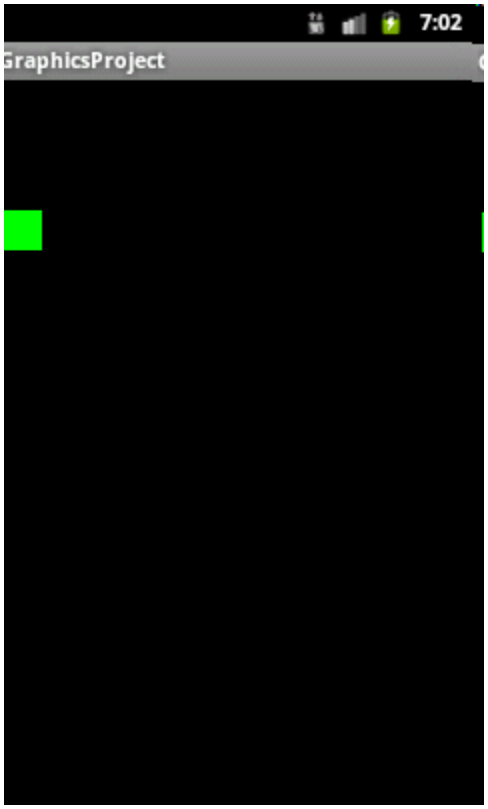
Multi-media Programming

- On a View
  - If your application does not require a significant amount of processing or frame-rate speed, then you should consider creating a custom View component and drawing with a Canvas in View.onDraw()
  - Each time that your application is prepared to be drawn, you must request your View be invalidated by calling invalidate()

Drawing graphics on a view

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mcv = new MyCustomView(this);
    setContentView(mcv);
}
```

```java
public class MyCustomView extends View {

    private ShapeDrawable sd;
    private int x = 50, y = 50, width = 20, height = 20;

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }

    public MyCustomView(Context context) {
        super(context);
        sd = new ShapeDrawable(new RectShape());
        sd.getPaint().setColor(0xff00ff00);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        sd.setBounds(x, y, x + width, y + height);
        sd.draw(canvas);
    }

}
```



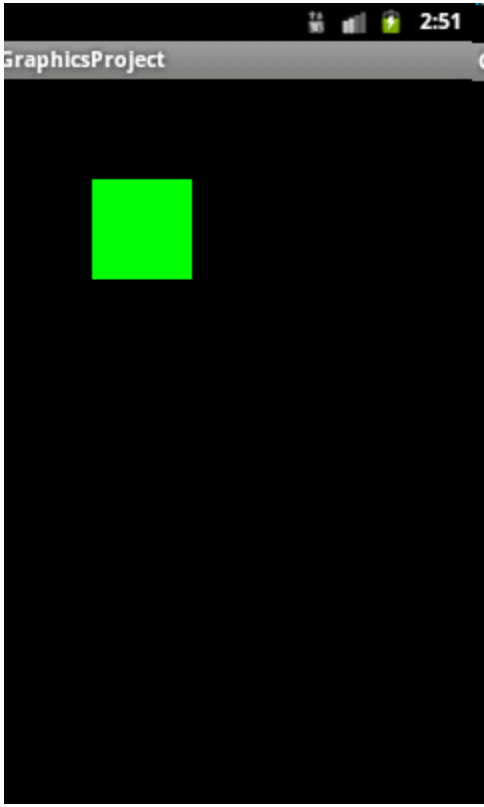
Multi-media Programming

- On a SurfaceView
  - The SurfaceView is a special subclass of View that offers a dedicated drawing surface within the View hierarchy
  - To begin, you need to create a new class that extends SurfaceView. The class should also implement SurfaceHolder.Callback
  - Instead of handling the Surface object directly, you should handle it via a SurfaceHolder

Drawing graphics on a SurfaceView

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MySurfaceView msv = new MySurfaceView(this);
    setContentView(msv);
}
```

```java
public class MySurfaceView extends SurfaceView implements Callback {
    SurfaceHolder sh;

    public MySurfaceView(Context context) {
        super(context);
        sh = this.getHolder();
        sh.addCallback(this);
    }

    @Override
    public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {
        // TODO Auto-generated method stub

    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        // TODO Auto-generated method stub
        holder.addCallback(this);
        Canvas c = holder.lockCanvas();
        Paint p = new Paint();
        p.setColor(0xff00ff00);
        c.drawRect(50, 50, 100, 100, p);
        holder.unlockCanvasAndPost(c);
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        // TODO Auto-generated method stub
    }
}
```

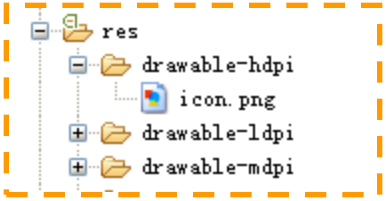


cn/~waterzhj

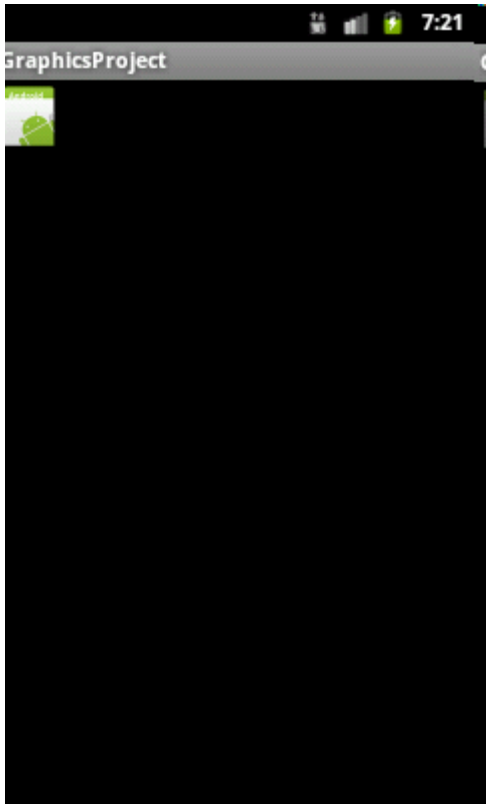
Multi-media Programming

- Drawables
  - Creating from resource image
    - Supported file types are PNG (preferred), JPG (acceptable) and GIF (discouraged)
  - Creating from resource XML
    - You can define your drawable in XML
  - Shape Drawable
    - With a ShapeDrawable, you can programmatically draw primitive shapes and style them in any way imaginable

Creating Drawable from resource image

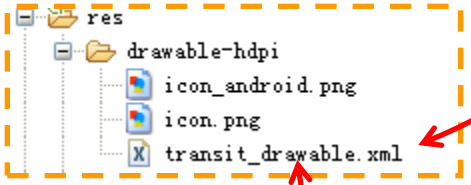



```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    LinearLayout ll = new LinearLayout(this);
    ImageView iv = new ImageView(this);
    Resources res = this.getResources();
    iv.setImageDrawable(res.getDrawable(R.drawable.icon));
    iv.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT));
    iv.setAdjustViewBounds(true);
    ll.addView(iv);
    setContentView(ll);
}
```

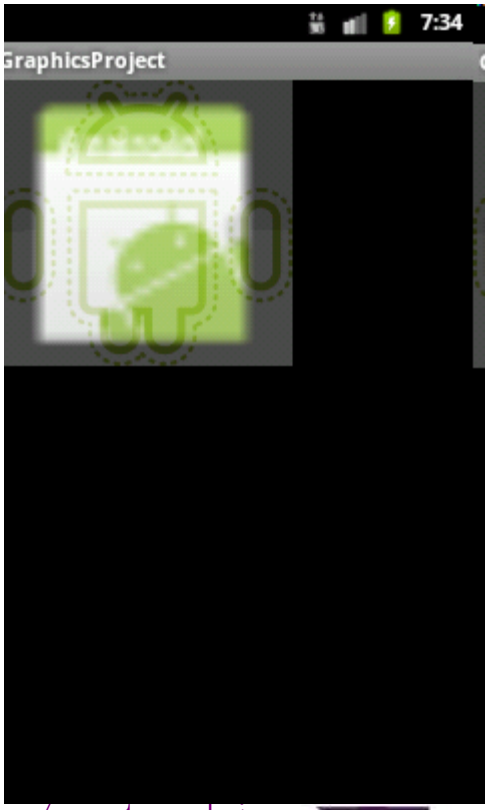


Creating Drawable from resource XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/icon"></item>
    <item android:drawable="@drawable/icon_android"></item>
</transition>
```




```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    LinearLayout ll = new LinearLayout(this);
    ImageView iv = new ImageView(this);
    Resources res = this.getResources();
    TransitionDrawable td = (TransitionDrawable) res
            .getDrawable(R.drawable.transit_drawable);
    iv.setImageDrawable(td);
    iv.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT));
    iv.setAdjustViewBounds(true);
    ll.addView(iv);
    setContentView(ll);
    td.startTransition(3000);
}
```



Creating Shape Drawable

```
public class CustomDrawableView extends View {
private ShapeDrawable mDrawable;

public CustomDrawableView(Context context) {
super(context);

int x = 10;
int y = 10;
int width = 300;
int height = 50;

mDrawable = new ShapeDrawable(new OvalShape());
mDrawable.getPaint().setColor(0xff74AC23);
mDrawable.setBounds(x, y, x + width, y + height);
}

protected void onDraw(Canvas canvas) {
mDrawable.draw(canvas);
}
}
```

Multi-media Programming

- OpenGL is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware
- Android includes support for high performance 2D and 3D graphics with the Open Graphics Library (OpenGL) , specifically, the OpenGL ES API

Multi-media Programming

- Android supports OpenGL both through its framework API and the Native Development Kit (NDK)
- There are two foundational classes in the Android framework that let you create and manipulate graphics with the OpenGL ES API
  - GLSurfaceView
  - GLSurfaceView.Renderer

Multi-media Programming

- View Animation
  - You can use the view animation system to perform tweened animation on Views
    - Tween animation calculates the animation with information such as the start point, end point, size, rotation, and other common aspects of an animation
  - A sequence of animation instructions defines the tween animation, defined by either XML or Android code

Multi-media Programming

- View Animation (cont.)
  - The view animation framework supports both tween and frame by frame animations, which can both be declared in XML
  - Tween Animation
    - An animation defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic

Multi-media Programming

- Tween Animation (cont.)
  - The XML file must have a single root element: either an <alpha>, <scale>, <translate>, <rotate>, or <set> element that holds a group (or groups) of other animation elements
  - <set>
    - A container that holds other animation elements (<alpha>, <scale>, <translate>, <rotate>) or other <set> elements
  - <alpha>
    - A fade-in or fade-out animation

Multi-media Programming

- Tween Animation (cont.)
  - <scale>
    - A resizing animation
  - <translate>
    - A vertical and/or horizontal motion
  - <rotate>
    - A rotation animation
  - Interpolators
    - An interpolator is an animation modifier defined in XML that affects the rate of change in an animation

```xml
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <scale
        android:interpolator="@android:anim/accelerate_decelerate_interpolator"
        android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="false"
        android:duration="700" />
    <set
        android:interpolator="@android:anim/accelerate_interpolator"
        android:startOffset="700">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.0"
            android:fromYScale="0.6"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="400" />
    </set>
</set>
```

Tween Animation

```java
public class AnimationProjectActivity extends Activity {
    /** Called when the activity is first created. */
    private TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.text);
        Animation ani = AnimationUtils.loadAnimation(this, R.anim.animate1);
        tv.startAnimation(ani);
    }
}
```

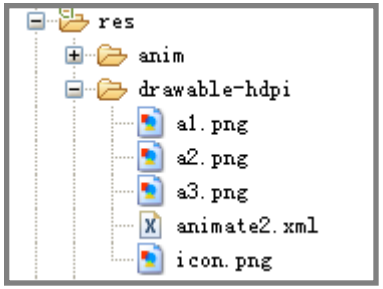
```xml
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">
    <alpha android:fromAlpha="0.0" android:toAlpha="1.0"
        android:duration="3000" android:interpolator="@android:anim/linear_interpolator"></alpha>
    <translate android:fromXDelta="0.0" android:toXDelta="50.0"
        android:fromYDelta="0.0" android:toYDelta="50.0" android:duration="3000"
        android:interpolator="@android:anim/linear_interpolator">
    </translate>
</set>
```

Multi-media Programming

- Frame animation
  - An animation defined in XML that shows a sequence of images in order
  - The XML file must have a single root element: <animation-list>
  - <animation-list>
    - This must be the root element. Contains one or more <item> elements
  - <item>
    - A single frame of animation. Must be a child of a <animation-list> element

Frame Animation

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/a1" android:duration="1000"></item>
    <item android:drawable="@drawable/a2" android:duration="1000"></item>
    <item android:drawable="@drawable/a3" android:duration="1000"></item>
</animation-list>
```

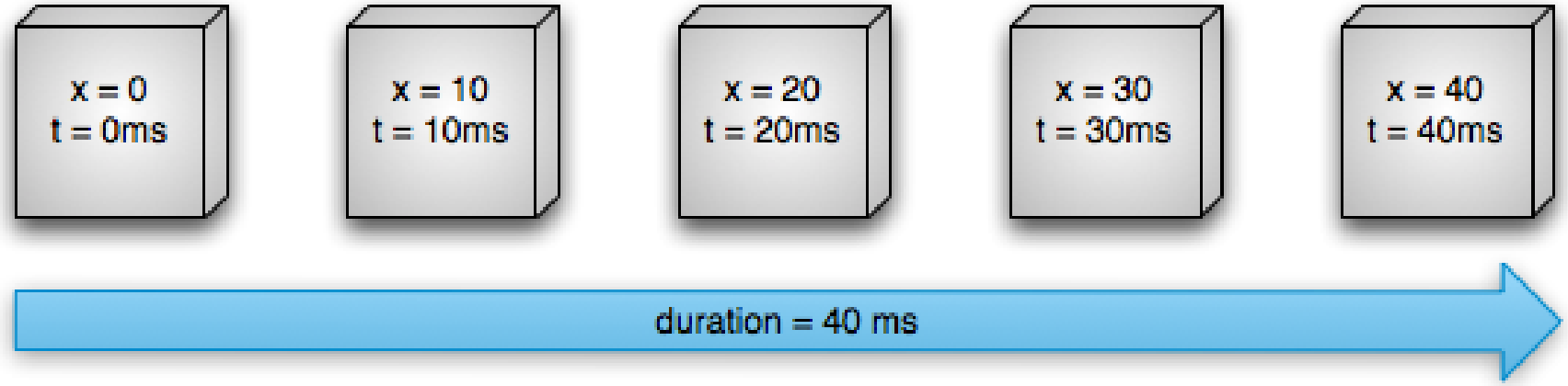



```
public class AnimationProjectActivity extends Activity {
    /** Called when the activity is first created. */
    private ImageView iv;
    AnimationDrawable ad;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        iv = (ImageView) findViewById(R.id.img);
        iv.setBackgroundResource(R.drawable.animate2);
        ad = (AnimationDrawable) iv.getBackground();
    }

    public void clickButton(View v) {
        if (v.getId() == R.id.click) {
            ad.start();
        }
    }
}
```

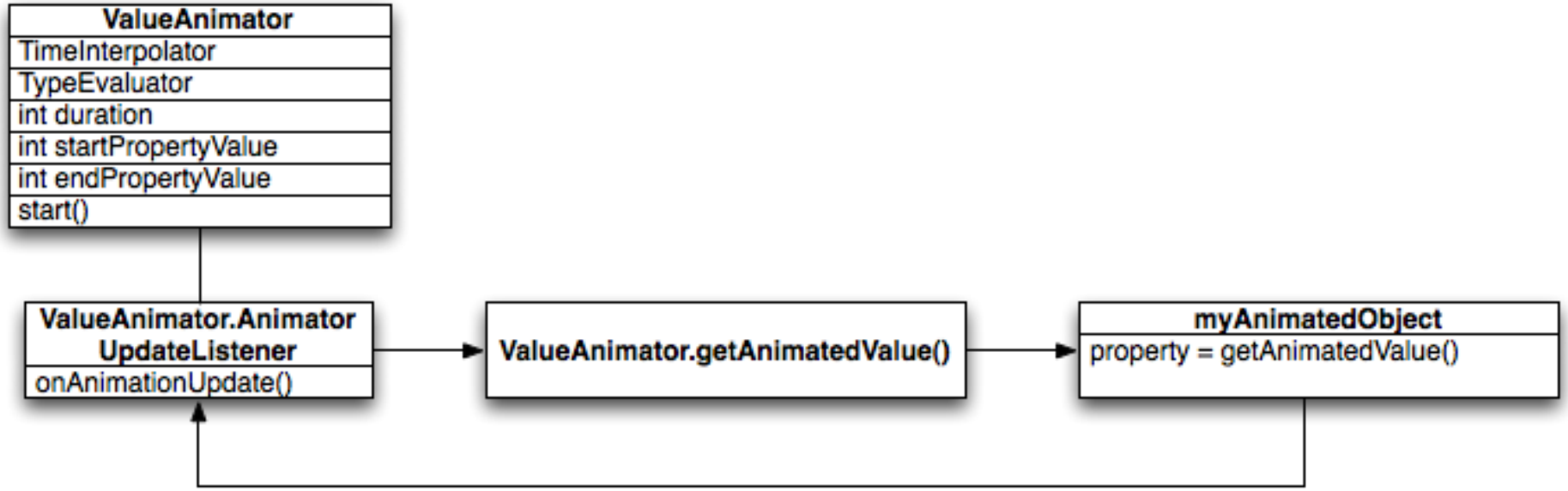
It's important to note that the start() method cannot be called during the onCreate() method of your Activity

Multi-media Programming

- Property Animation
  - You can define an animation to change any object property over time
    - A property animation changes a property's (a field in an object) value over a specified length of time
  - The property animation system lets you define the following characteristics of an animation
    - Duration, time interpolation, repeat count and behavior, animator sets, frame refresh delay



Example of a linear animation



How animations are calculated

Multi-media Programming

- ValueAnimator
  - provides a simple timing engine for running animations which calculate animated values and set them on target objects
  - By default, ValueAnimator uses non-linear time interpolation

```
<animator xmlns:android="http://schemas.android.com/apk/res/android"
          android:duration="1000"
          android:repeatCount="1"
          android:repeatMode="reverse">
    <propertyValuesHolder>
        <keyframe android:fraction="0" android:value="1"/>
        <keyframe android:fraction=".2" android:value=".4"/>
        <keyframe android:fraction="1" android:value="0"/>
    </propertyValuesHolder>
</animator>
```

Multi-media Programming

- ObjectAnimator
  - provides support for animating properties on target objects
  - To use ObjectAnimator, you must do the following
    - To instantiate an ObjectAnimaor, you need to specify the object and the name of that object's property along with the values to ainimate between
    - The object's property animated must have a setter method
    - The getter and setter methods of the property animated must operate on the same type as the starting and ending values
    - You need to call the invalidate() method on a View to force redraw it

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvyY).start();
```

Example of ObjectAnimator

Multi-media Programming

- Drawable Animation
  - Drawable animation lets you load a series of Drawable resources one after another to create an animation
    - you can define the frames of an animation in your code or xml file
  - The start() method cannot be called during the onCreate() method of your activity
    - because the AnimationDrawable is not yet fully attached to the window

Sample Drawable Animation

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

```
AnimationDrawable rocketAnimation;

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

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
  if (event.getAction() == MotionEvent.ACTION_DOWN) {
    rocketAnimation.start();
    return true;
  }
  return super.onTouchEvent(event);
}
```

Conclusions

- Structured & OO Programming
- Programming for Performance
- File Programming
- Network Programming
- Multi-media Programming
- Conclusions