

**Next**

Traccia: S2

Gruppo: T6

Commento e Codice

**Modulo: Phone**

MessageService

Modulo: Phone

public class MessageService extends WearableListenerService {  
 Intent messageIntent;  
  
 String attendancePath = "/attendance";  
 String payloadName = "attendance";  
  
 String refreshPath = "/refreshSchedule";

public MessageService() {  
 }  
  
 @Override  
 public void onMessageReceived(MessageEvent messageEvent){  
 if (messageEvent.getPath().equals(attendancePath)){  
  
 final String message = new String(

messageEvent.getData());  
 messageIntent= new Intent();  
 messageIntent.setAction(Intent.*ACTION\_SEND*);  
 messageIntent.putExtra(payloadName, message);  
  
 LocalBroadcastManager.*getInstance*(this)

.sendBroadcast(messageIntent);  
  
  
 } else if (messageEvent.getPath().equals(refreshPath)) {  
 //*TODO : implementare richiesta di aggiornamento* } else{  
 super.onMessageReceived(messageEvent);  
 }  
 }  
}

MessageService è una classe figlia di Wearable- ListenerService. La sua funzione è gestire la ricezione e il filtraggio di MessageEvents mandati dal modulo Wearable

attendancePathe **refreshPath** sono stringhe uniche che identificano i messaggi e permettono di classificarli e richiamare i rispettivi Intent. **payloadName** è il ‘carico’ che viene trasportato dai messaggi di attendance.

Alla ricezione del messageEvent si confronta il path contenuto in esso con i path sopra presentati.

Se il path ricevuto è **attendancePath ->**Viene chiamata un’ istanza di LocalBroadcast- Manager, che lancia un broadcast all’activity attiva.

Se il path ricevuto è **refreshPath ->**Un intent viene lanciato a Phone\_Activity\_Main con la richiesta di mandare un oggetto Schedule contenente la lista di lezioni odierne.

Phone\_Activity\_AddNewSchedule

Modulo: Phone

public class Phone\_Activity\_AddNewSchedule extends  
 AppCompatActivity {  
 private String[] CdL; //TODO ottenerle da internet  
 private String[] years; //TODO ottenerle da CdL  
 final String pathName = getApplicationContext()  
 .getFilesDir() + "next";  
 final InputStream tempResource = getResources()  
 .openRawResource(R.raw.schedule\_stubs);  
  
//TODO implementare priorità di inserimento(cdl-anni)  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(  
 R.layout.phone\_activity\_addnewschedule);

final TextView textField = findViewById(  
 R.id.nameFileField);  
 Button button = findViewById(  
 R.id.createButton);  
 button.setOnClickListener(  
 new View.OnClickListener() {  
  
 @Override  
 public void onClick(View v) {  
 if (textField.getText() != null) {  
 String fileName= textField.getText();  
 createfile(tempResource, fileName);  
 try {  
 tempResource.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 Toast.makeText(  
 getApplicationContext(),

"File created",

Toast.LENGTH\_SHORT).show();

Intent intent = new Intent(getBaseContext(),   
 Phone\_Activity\_Main.class);  
 intent.putExtra("REFRESH", true);  
 startActivity(intent);

} else {

Toast.makeText( getApplicationContext(),  
 "Insert file name",   
 Toast.LENGTH\_SHORT).show();  
 }  
  
 }  
 });  
 }

**Phone\_**Activity**\_AddNewSchedule** è una Activity.

La sua funzione è rispondere alle interazioni dell’utente sul layout **phone\_activity\_addnew-schedule** per creare nuovi file e salvarli all’interno della memoria dispositivo.

**CdL** e **years** sono entrambe Array di stringhe atte a memorizzare gli input ottenuti dal web rispettivamente: corsi di laura e quanti anni serve conseguire ciascuno di questi.

textField è il campo in cui l’utente immette il nome del file che vuole creare.

button è il pulsante che crea il file e reindirizza alla main activity.

Viene impostato un OnClickListener su button, questo verifica che alla pressione del pulsante ci sia un testo immesso in textField, finalizza quel valore e richiama la funzione **createfile**.

Il Toast rimanda un feedback all’utente, indicando che il file è stato creato o se manca il nome dello stesso. In futuro dovranno essere riempiti anche i campi di corso di laurea e anni.

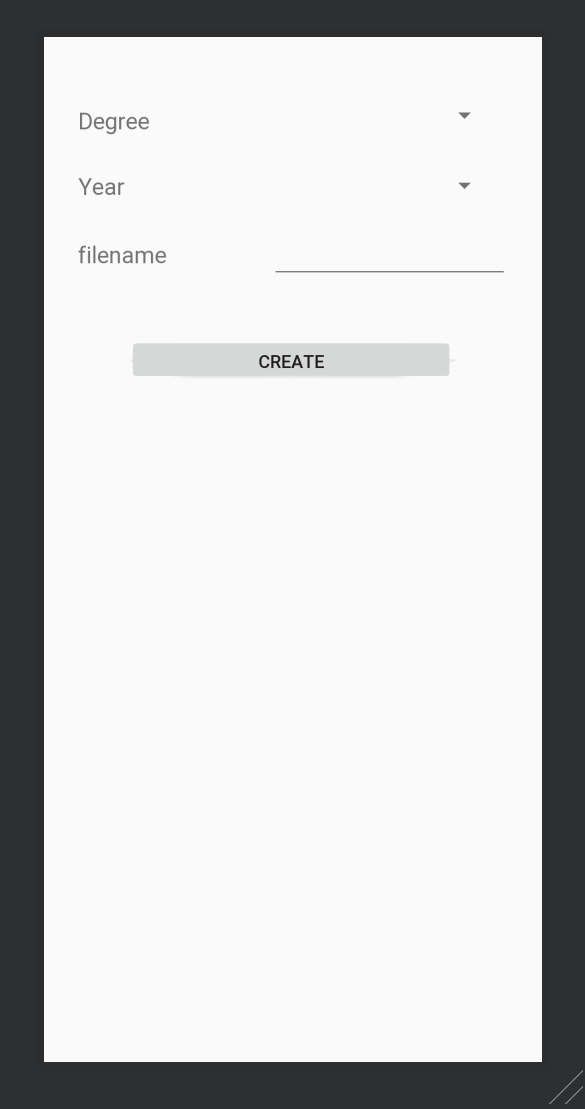
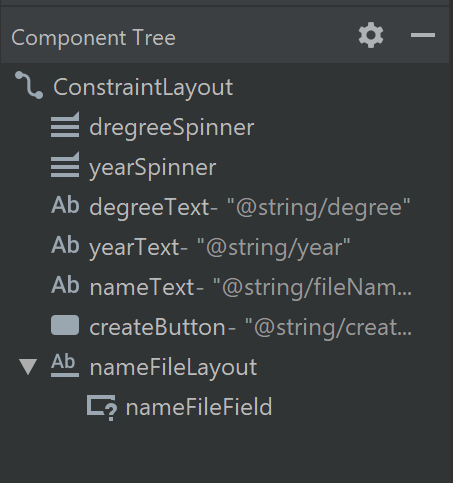
Viene creato un intent verso la classe **Phone\_Activity\_Main**  di aggiornare il frammento di visualizzazione lista orari e quindi fare un cambio di activity.

private void createfile(InputStream iStream, String fileName){  
 File pathFile = new File(pathName);  
 FileOutputStream oStream;  
 fileName = pathName + File.*separator* + fileName + ".xml";  
   
 if (!pathFile.isDirectory()){  
 pathFile.mkdir();  
 }  
   
 try {  
 File newFile = new File(fileName);  
 if(!newFile.isFile()) {  
 newFile.createNewFile();  
  
 oStream = new FileOutputStream(newFile);  
 IOUtils.*copyStream*(iStream, oStream);  
  
 oStream.flush();  
 oStream.close();  
 } else {  
 Toast.*makeText*(getApplicationContext(), "File   
 already existing, try another name",   
 Toast.*LENGTH\_SHORT*);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}  
  
class DownloadFileFromURL extends AsyncTask<String, String, String> {  
  
 @Override  
 protected String doInBackground(String[] f\_url) {  
 int count;  
 try {  
 URL url = new URL(f\_url[0]);  
 url.openConnection().connect();  
  
 InputStream input = new   
 BufferedInputStream(url.openStream(),  
 8192);  
 //*TODO salvare input* input.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return null;  
 }  
  
  
}

**Createfile** verifica che sia presente nel dispositivo la cartella ‘next’ nella giusta directory, se assente la crea. Procede poi a verificare la presenza del file nella suddetta, se presente restituisce tramite Toast il messaggio “file già esistente” suggerendo di cambiare il nome.

La creazione del file consiste nel flusso di dati tra inputStream e outputStream (il file stesso) e la chiusura di questi flussi.

**DownloadFileFromURL** eredita i metodi di una Task asincrona così da non dover interrompere l’interfaccia mentre è in funzione. Serve a scaricare in background in corrispondenza di un URL dei file. Verrà implementata meglio restituendo i file e parametri scaricati dal sito ufficiale edll’orario del PoliBa.

I Field si devono schiarire uno alla volta: l’utente deve prima inserire il Degree, poi l’Year, infine il filename e solo allora il pulsante ‘Create’ sarà utilizzabile.

Phone\_Activity\_DailySchedulePager

Modulo: Phone

public class Phone\_Activity\_DailySchedulePager extends FragmentActivity {  
 String dayText;  
 private static final int *NUM\_PAGES* =5;  
 private ViewPager mPager;  
 private PagerAdapter pagerAdapter;  
 private Schedule schedule;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*phone\_activity\_schedulepager*);  
 Bundle scheduleBundle = getIntent().getExtras();

mPager = findViewById(R.id.*pager*);  
 pagerAdapter = new   
 ScreenSlidePagerAdapter(getSupportFragmentManager());  
 mPager.setPageTransformer(true, new   
 ZoomOutPageTransformer());  
 mPager.setAdapter(pagerAdapter);  
   
 TextView dayTextField = findViewById(R.id.*dayText*);  
 dayTextField.setText(dayText);  
 }

@Override  
 public void onBackPressed() {  
 if (mPager.getCurrentItem() == 0) {  
 super.onBackPressed();  
 } else {  
 mPager.setCurrentItem(mPager.getCurrentItem() - 1);  
 }  
 }  
   
 public void setDay(int d){  
 switch (d){  
 case (Calendar.*TUESDAY*):  
 dayText = getString(R.string.*tuesday*);  
 break;  
 case (Calendar.*WEDNESDAY*):  
 dayText = getString(R.string.*wednesday*);  
 break;  
 case (Calendar.*THURSDAY*):  
 dayText = getString(R.string.*thursday*);  
 break;  
 case (Calendar.*FRIDAY*):  
 dayText = getString(R.string.*friday*);  
 break;  
 default:  
 dayText = getString(R.string.*monday*);  
 break;  
 }  
 }

**Phone\_Activity\_DailySchedulePager** è un’Acti-vity che eredita I metodi di FragmentActivity. La sua funzione è di gestire e rispondere alle intera-zioni utente all’interfaccia ***phone\_activity\_sche-dulepager.*** Gestisce i passaggi e le animazioni tra i Fragment **Phone\_Fragment\_DailySchedule** tramite ViewPager.

***NUM\_PAGES*** indica il numero di frammenti tra cui poter scorrere (5 poiché il calendario settimanale va da lunedì a venerdì).

Quando viene creata l’Activity chiede i dati con cui è stata chiamata e li salva nel Bundle **scheduleBundle**, questo permette di aggiornare i valori interni (per essere implementato la classe Stub deve ereditare i metodi di Parceable).

**mPager** viene istanziato e impostato come FragmentManager, viene impostata anche l’animazione ZoomOutTransformer poi definita.

**dayTextField** è l’indicatore della giornata corrispondente al Fragment, per lo stato attuale di implementazione non viene aggiornato.

onBackPressed viene eseguita quando l’utente utilizza il pulsante di default per tornare indietro, e fa scorrere indietro nella settimana. In futuro verrà eliminato.

setDay imposta la variabile **dayText** in base al valore int d in input.

private class ScreenSlidePagerAdapter extends   
 FragmentStatePagerAdapter {  
 public ScreenSlidePagerAdapter(FragmentManager fm) {  
 super(fm);  
 }  
  
 @NonNull  
 @Override  
 public Fragment getItem(int position) {  
 return new Phone\_Fragment\_DailySchedule();  
 }  
  
 @Override  
 public int getCount() {  
 return *NUM\_PAGES*;  
 }  
 }

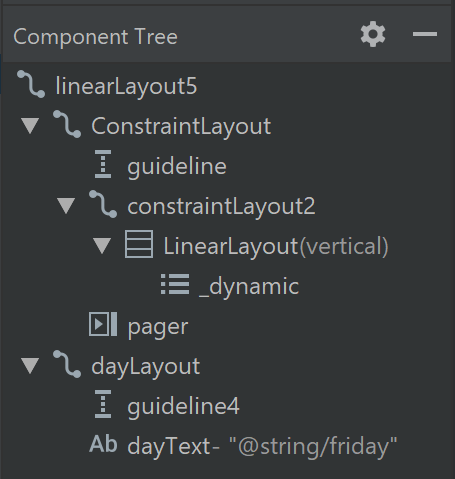
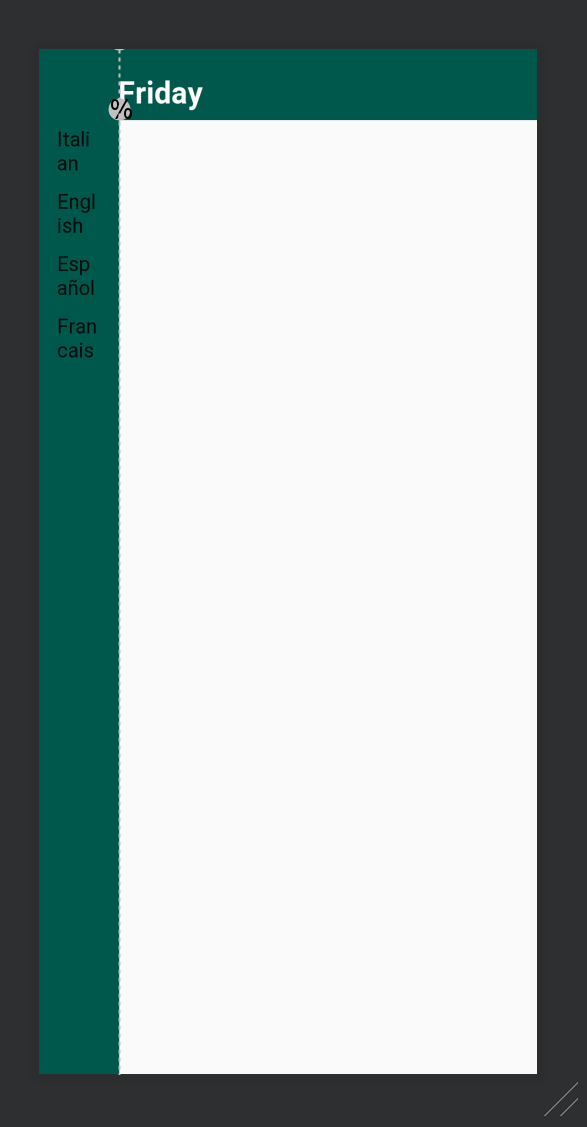
private class ZoomOutPageTransformer implements   
 ViewPager.PageTransformer {  
 private static final float *MIN\_SCALE* = 0.85f;  
 private static final float *MIN\_ALPHA* = 0.5f;  
  
 public void transformPage(View view, float position) {  
 int pageWidth = view.getWidth();  
 int pageHeight = view.getHeight();  
  
 if (position < -1) { // [-Infinity,-1)  
 // This page is way off-screen to the left.  
 view.setAlpha(0f);  
  
 } else if (position <= 1) { // [-1,1]  
 float scaleFactor =   
 Math.*max*(*MIN\_SCALE*, 1 - Math.*abs*(position));  
 float vertMargin =   
 pageHeight \* (1 - scaleFactor) / 2;  
 float horzMargin =   
 pageWidth \* (1 - scaleFactor) / 2;  
 if (position < 0) {  
 view  
 .setTranslationX( horzMargin - vertMargin /2);  
 } else {  
 view  
 .setTranslationX(-horzMargin + vertMargin /2);  
 }  
  
 // Scale the page down (between MIN\_SCALE and 1)  
 view.setScaleX(scaleFactor);  
 view.setScaleY(scaleFactor);  
  
 // Fade the page relative to its size.  
 view.setAlpha(*MIN\_ALPHA* +  
 (scaleFactor - *MIN\_SCALE*) /  
 (1 - *MIN\_SCALE*) \* (1 - *MIN\_ALPHA*));  
  
 } else { // (1,+Infinity]   
 view.setAlpha(0f);  
 }  
 }  
 }  
}

**ScreenSlidePagerAdapter** eredita i metodi di FragmentStatePagerAdapter per creare un pager efficace.

Il costruttore rimanda al costruttore della classe super.

getItem restituisce l’oggetto che viene mostrato tramite la sua posizione.

ZoomOutPageTransformer è un’animazione di cui non abbiamo credito:   
<https://gist.github.com/xalexchen/9423660>.



Phone\_Activity\_Main

Modulo: Phone

public class Phone\_Activity\_Main extends AppCompatActivity{  
  
 //*TODO richiedi permessi di storage SE non sono già concessi* Schedule currentSchedule;  
 protected Handler messageHandler;  
  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*phone\_activity\_main*);  
   
 Objects.*requireNonNull*(  
 this.getSupportActionBar()).hide();  
  
 messageHandler = new Handler(new Handler.Callback() {  
 @Override  
 public boolean handleMessage(@NonNull Message msg) {  
 Bundle stuff = msg.getData();  
 messageText(stuff.getString("messageText"));  
 return true;  
 }  
 });  
  
 LocalBroadcastManager  
 .*getInstance*(this).registerReceiver(  
 new Receiver(),  
 new IntentFilter(Intent.*ACTION\_SEND*)  
 );  
 }  
  
 //COMMUNICATION  
 public class Receiver extends BroadcastReceiver {

@Override  
 public void onReceive(Context context, Intent intent) {  
 String message =   
 "i received a message from the wearable";  
 Toast.*makeText*(  
 context, message, Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
  
 public void messageText(String message){  
 String path = "next";  
 new MessageThread(path,message).start();  
 }

**Phone\_Activity\_Main** è l’Activity principale. La sua funzione principale è ridirezionare l’utente tra le varie activity, gestire i file orari salvati e gestire i messaggi tra i moduli.

**currentSchedule** memorizza l’attuale oggetto Schedule.

**messageHandler** permette all’Activity di leggere messaggi ricevuti dal LocalBroadcastManager, di cui dopo si crea l’istanza e si impostano i filtri di intenti.

getSupportActionBar.hide è una funzione che abbiamo dovuto ripetere spesso nelle Activity per nascondere l’ActionBar di default.

**Receiver** eredita i metodi di BroadcastReceiver, serve a notificare l’utente tramite Toast quando viene ricevuto un messaggio dall’altro modulo.

messageText invia un messaggio sul path **next** al wearable, verrà cambiato con i reali Intent da mandare (risposta alla richiesta di aggiorna-mento e registro delle presenze) .

class MessageThread extends Thread{  
 String path;  
 String message;

MessageThread(String p, String m){  
 path = p;  
 message = m;  
 }  
  
 public void run(){  
 Task<List<Node>> wearableList =   
 Wearable.*getNodeClient*(getApplicationContext())

.getConnectedNodes();  
 try{  
 List<Node> nodes = Tasks.*await*(wearableList);  
 for (Node node : nodes){  
 Task<Integer> sendMessageTask =  
   
 Wearable.*getMessageClient*(Phone\_Activity\_Main.this)  
 .sendMessage(  
 node.getId(), path, message.getBytes());  
  
 try{  
 Tasks.*await*(sendMessageTask);  
 messageText(  
 "I sent a message to the wearable");  
 } catch (  
 ExecutionException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 } catch (

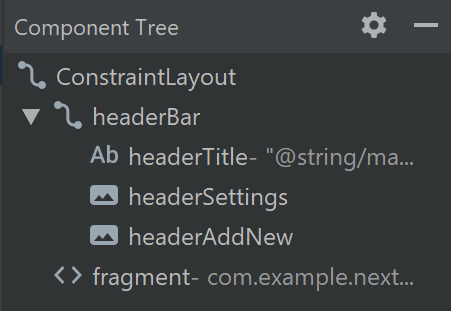
InterruptedException | ExecutionException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
  
 //BUTTONS  
 public void onClick\_addNew(View view){  
 startActivity(new Intent(this,   
 Phone\_Activity\_AddNewSchedule.class));  
 }  
 public void onClick\_settings(View view){  
 startActivity(new Intent (this,   
 Phone\_Activity\_Settings.class));  
 }  
 public void setCurrentSchedule(Schedule schedule) {  
 this.currentSchedule = schedule;  
 }  
 }

**MessageThread** eredita i metodi di Thread, la sua funzione è gestire i messaggi in parallelo alla normale esecuzione dell’applicazione:

il metodo run tramite getNodeClient ottiene la lista di Node (dispositivi) wearableList collegati e manda un messaggio verso tutti (in futuro verrà imposto che il Node sia solo di tipo Wearable) passando all’utente tramite Toast il feedback di l’avvenuta consegna (anche se non è confermata la ricezione).

onClick\_addNew e onClick\_settings creano entrambi Intent per chiamare le rispettive Activity(**Phone\_**Activity**\_AddNewSchedule** e **Phone\_Activity\_Settings**).

setCurrentSchedule è il metodo di scrittura usato poi dal Fragment **Phone\_Fragment-\_ScheduleList**.



La fascia bianca in alto è un piccolo problema da risolvere, in quanto la barra delle notifiche in alto copre parzialmente la barra del titolo

Phone\_Activity\_Settings

Modulo: Phone

public class Phone\_Activity\_Settings extends AppCompatActivity{  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*phone\_activity\_settings*);  
 this.getSupportActionBar().hide();  
  
 getSupportFragmentManager()  
 .beginTransaction()  
 .replace(R.id.*settings*, new SettingsFragment())  
 .commit();  
 }

public static class SettingsFragment extends   
 PreferenceFragmentCompat {

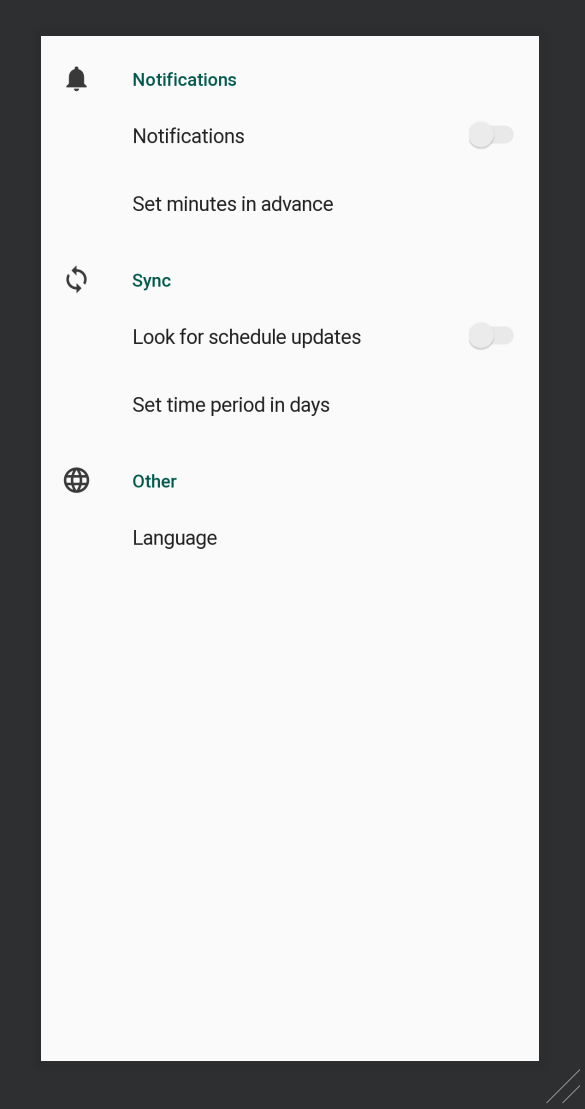
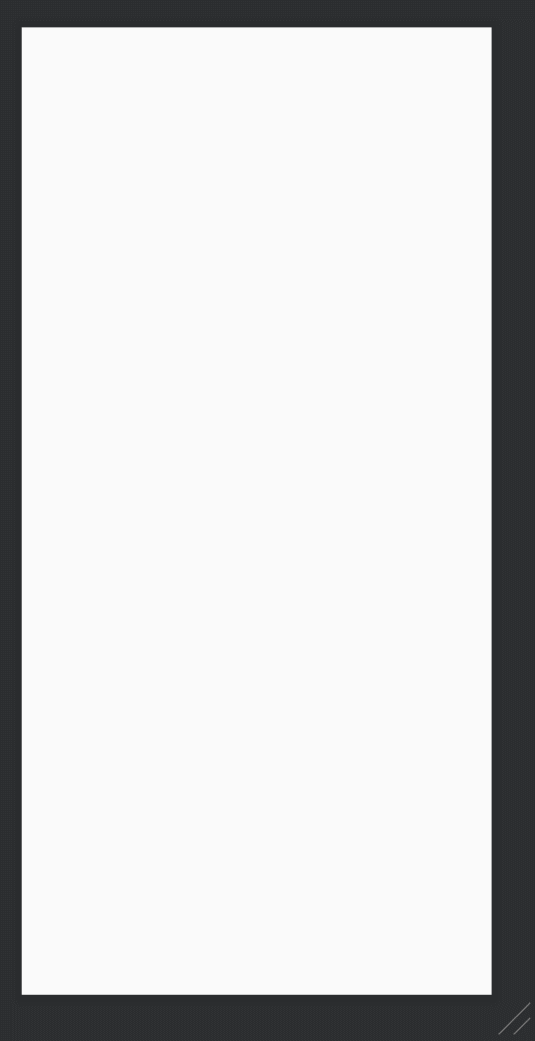
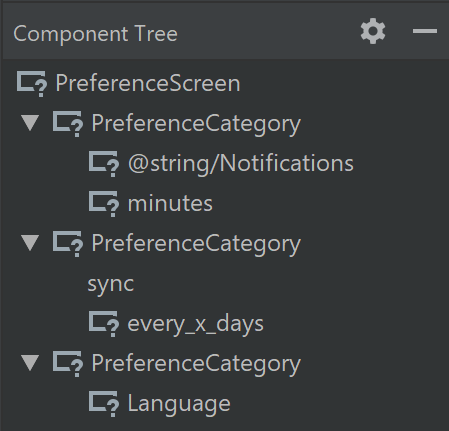
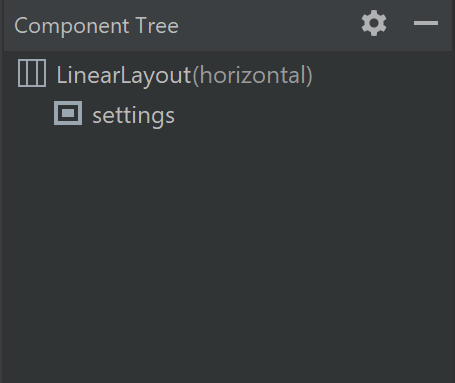
@Override  
 public void onCreatePreferences(

Bundle savedInstanceState, String rootKey)

{  
 setPreferencesFromResource(  
 R.xml.*root\_preferences*, rootKey);  
 }  
 }  
 }

**Phone\_Activity\_Settings** è l’Activity dedicata alla gestione da parte dell’utente delle proprie preferenze. È un pattern già definito da Android Studio, la utilizziamo in maniera tale da disporre un’interfaccia già famigliare all’utente.

Alla creazione il layout phone\_activity\_settingsviene sostituito con un SettingsFragment, che chiama a sua volta il layout root\_preference**.**

****

Phone\_Fragment\_DailySchedule

Modulo: Phone

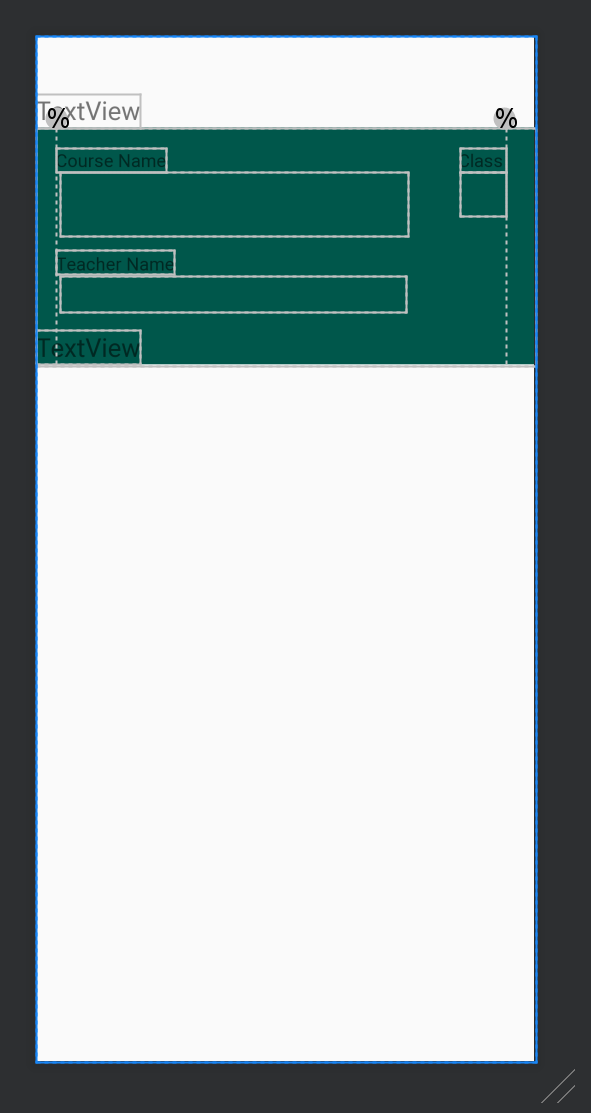
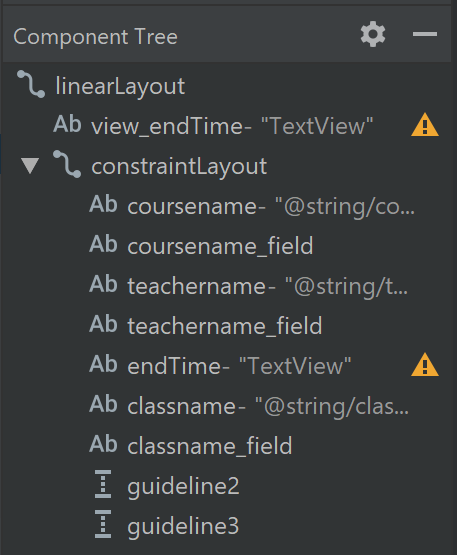
public class Phone\_Fragment\_DailySchedule extends Fragment {  
  
 @Override  
 public View onCreateView(  
 LayoutInflater inflater,   
 ViewGroup container,  
 Bundle savedInstanceState)

{

ViewGroup rootView = (ViewGroup) inflater.inflate(  
 R.layout.*phone\_fragment\_dailyschedule*,   
 container, false);  
  
 return rootView;  
 }  
 }

**Phone\_Fragment\_DailySchedule** è ancora da implementare, viene chiamata in Phone\_Acti-vity\_DailySchedulePager e servirà a suddividere in giornate lo Schedule e ad ordinare gli elementi visivi nel layout.

Dovrà disporre layout phone\_item\_dailystub in secondo l’orario e la giornata.



Il colore di sfondo al costraintLayout principale sarà randomicamente generato ma si manterrà su colori molto neutri e con scritte bianche,

gli attuali errori sono dovuti a HardText impostato nei Field

Phone\_Fragment\_ScheduleList

Module: Phone

public class Phone\_Fragment\_ScheduleList extends   
 ListFragment{  
 private File folder;

@Override  
 public View onCreateView(LayoutInflater inflater, ViewGroup   
 container, Bundle savedInstanceState){  
 String path = Objects.*requireNonNull*(  
 getContext()).getFilesDir() + "next";

if (folder == null)  
 folder = new File(path);

return inflater.inflate(  
 R.layout.*phone\_fragment\_schedulelist*,

container, false);  
 }  
  
  
 @Override  
 public void onActivityCreated(

@Nullable Bundle savedInstanceState){  
 super.onActivityCreated(savedInstanceState);  
 String[] filenames;  
 filenames = getFilenames();

if (filenames != null) {  
 CustomListAdapter cListAdapter =

new CustomListAdapter(  
 getActivity(), getFilenames());  
 setListAdapter(cListAdapter);  
 }  
 }

private String[] getFilenames(){  
 File[] files = folder.listFiles();  
 if (files == null)  
 return null;  
  
 String[] filenames = new String[files.length];  
 for (int i=0; i< files.length; i++){  
 filenames[i] = files[i].getName();  
 }  
  
 return filenames;  
 }

**Phone\_Fragment\_ScheduleList** gestisce la lista di file Schedule nel path di storage interno, associato al File folder.

onCreateView si assicura che la directory esista e tramite floater apre il layout di phone\_frag-ment\_schedulelist**.**

Alla creazione dell’Activity genitrice (Phone\_Ac-tivity\_Main), onActivityCreated crea un’istanza di CustomListAdapter che inserisce la lista di file nella cartella inphone\_fragment-\_schedulelist, lista ottenuta dal metodo getFilenames.

getFilenames richiama il vettore di File files presenti in folder, se questo vettore non è nullo viene scorso in un ciclo *for* insieme ad un vettore di stringhe appena creatofilenames, inserendo in filenames[i]il valore di files[i]; se il vettore files è vuoto viene restituito un vettore vuoto.

//*TODO cambia in recyclerView* class CustomListAdapter extends ArrayAdapter {  
 private Activity context;  
 private String[] nameArray;  
  
 CustomListAdapter(Activity context, String[] nameArray){  
 super(context, *phone\_item\_schedule*, nameArray);  
 this.context=context;  
 this.nameArray = nameArray;  
 }

@SuppressLint({"ViewHolder","InflateParams"})  
 @Override  
 public View getView(

int position, View view, ViewGroup parent) {

TextView scheduleNameField;  
 final ImageButton optionsButton;  
 final Schedule schedule =   
 new Schedule(Objects.*requireNonNull*(  
 folder.listFiles())[position]);

LayoutInflater inflater;  
 inflater = context.getLayoutInflater();  
 View itemView =inflater.inflate(

*phone\_item\_schedule*, null,true);  
  
 scheduleNameField = itemView.findViewById(

R.id.*filename*);  
 scheduleNameField.setText(nameArray[position]);  
 scheduleNameField.setOnClickListener(

new View.OnClickListener() {

@Override  
 public void onClick(View v) {  
 ((Phone\_Activity\_Main)   
 Objects.*requireNonNull*(getActivity())).  
 setCurrentSchedule(schedule);

Toast.*makeText*(getContext(),  
 "Set as current schedule",   
 Toast.*LENGTH\_SHORT*).show();  
 }  
 });

optionsButton = itemView.findViewById(  
 R.id.*popupmenuOpener*);  
  
 optionsButton.setOnClickListener(  
 new View.OnClickListener(){

@Override  
 public void onClick(View v){  
 final PopupMenu popupMenu =   
 new PopupMenu(  
 getContext(), optionsButton );  
 popupMenu.getMenuInflater().inflate(  
 R.menu.*schedule\_popup\_menu*,   
 popupMenu.getMenu());

La classe **CustomListAdapter** eredita i metodi di ArrayAdapter, viene istanziata per ogni File nel path dell’applicazione.

Ogni elemento nella lista è composto a sua volta da due elementi: il TextView scheduleNameField e l’ImageButton optionsButton.

Il testo impostato su scheduleNameField viene ottenuto in base alla position che assume nella lista (in questo modo si associa un file al nome dello stesso in due vettori diversi).

Viene creato un OnClickListener su scheduleNa-meField che se interpellato richiama la funzione setCurrentSchedule della Phone\_Activity\_Main impostando come schedule attuale quella associata alla position dell’item.

Su optionsButton viene impostato OnClickLis-tener che quando interpellato crea il menu popup schedule\_popup\_menu dal quale l’utente può scegliere tra varie opzioni, ognuna chiama un metodo di tipo void non ancora implementato.

popupMenu.setOnMenuItemClickListener(

new PopupMenu.

OnMenuItemClickListener() {  
 @Override  
 public boolean onMenuItemClick(

MenuItem item) {  
 switch (item.getItemId()){  
 case R.id.*view*:  
 view(schedule);  
 case R.id.*edit*:  
 edit(schedule);  
 case R.id.*rename*:  
 rename(schedule);  
 case R.id.*delete*:  
 delete(schedule);  
 case R.id.*refresh*:  
 refresh(schedule);  
 case R.id.*share*:  
 share(schedule);  
 }  
 return false;  
 }  
 });  
 popupMenu.show();  
 }  
 });  
 return itemView;  
 }  
 }

//MENU OPTIONS  
 Intent intent;

private void view(Schedule schedule){  
 intent = new Intent(getContext(),   
 Phone\_Activity\_DailySchedulePager.class);  
 intent.putExtra("SCHEDULE", schedule.getSchedule());  
 startActivity(intent);  
 }  
 private void edit(Schedule schedule){  
 //*TODO : Implement edit* }  
 private void rename(Schedule schedule){  
 //*TODO : Implement rename* }  
 private void delete(Schedule schedule){  
 //*TODO : Implement delete* }  
 private void refresh(Schedule schedule){  
 //*TODO : Implement refresh* }  
 private void share(Schedule schedule){  
 //*TODO : Implement share* }  
}

Ogni metodo agisce sullo Schedule restituito dal menu popup:

**view** ne permette la visualizzazione;

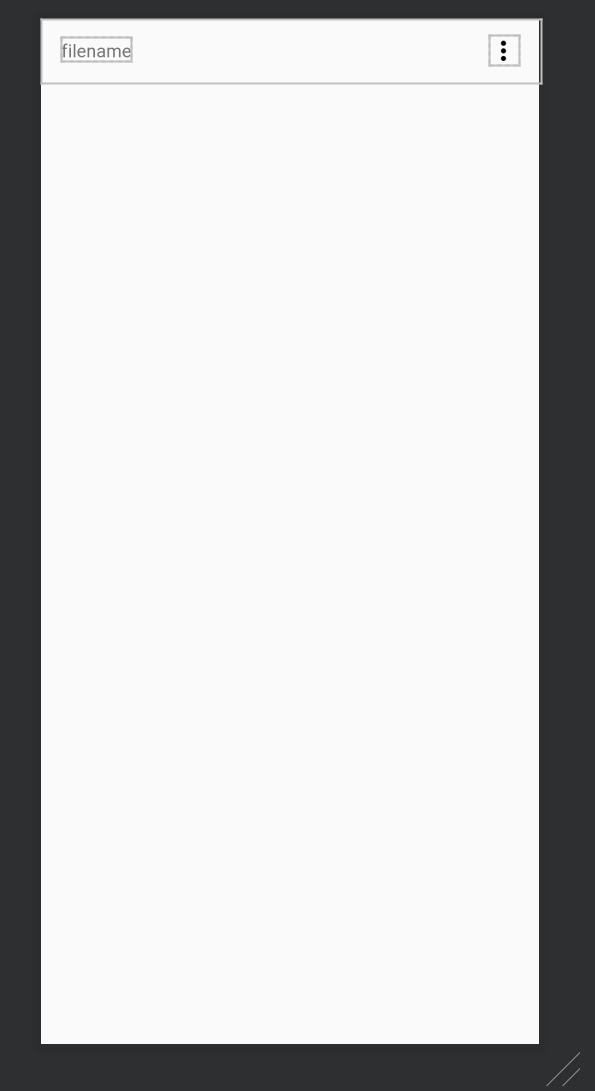
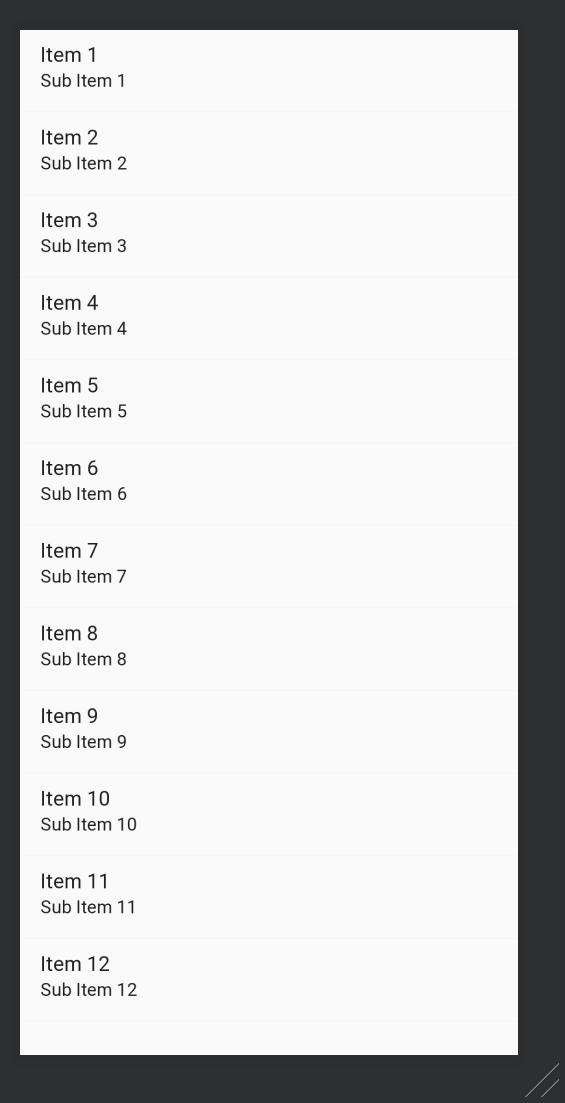
**edit** ne permette la modifica;

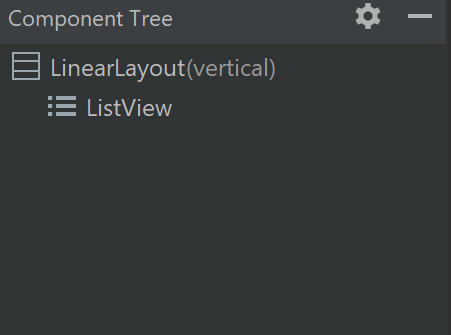
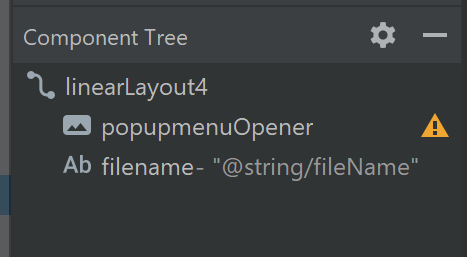
**rename** ne permette la rinominazione;

**delete** ne permette l’eliminazione;

**refresh** forza l’aggiornamento del file;

**share** lancia il menu di condivisione Android.




Commento e Codice

**Modulo: MyLibrary**

Schedule

Module: MyLibrary

public class Schedule {  
 private ArrayList<Stub> schedule;  
  
 public Schedule(File file) {  
 InputStream input = null;  
 try {  
 input = new FileInputStream(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 parseXML(input);  
 }  
  
 public Schedule(ArrayList<Stub> schedule){  
 this.schedule = schedule;  
 }  
  
 public Schedule(){  
 }  
  
 public ArrayList<Stub> getSchedule() {  
 return schedule;  
 }  
  
 //PARSING  
 private void parseXML(InputStream istream) {  
 if (schedule != null)  
 schedule.clear();  
 else  
 schedule = new ArrayList<>();  
 XmlPullParserFactory factory;  
 String text = "";  
 Stub stub= new Stub();  
 XmlPullParser parser;  
 int eventType;  
  
 try {  
 factory = XmlPullParserFactory.*newInstance*();  
 factory.setNamespaceAware(true);  
  
 parser = factory.newPullParser();  
 parser.setInput(istream, null);  
  
 eventType = parser.getEventType();  
  
 while (eventType != XmlPullParser.*END\_DOCUMENT*) {  
 String tagname = parser.getName();  
  
 switch (eventType) {  
 case XmlPullParser.*START\_TAG*:  
 if (tagname.equalsIgnoreCase("course")) {  
 // create a new instance of Stub  
 stub = new Stub();  
 }  
 break;  
  
 case XmlPullParser.*TEXT*:  
 text = parser.getText();  
 break;

La classe **Schedule** consiste principalmente nell’ArrayList di Stub **schedule** e nella loro manipolazione.

Il suo metodo principale è parseXML: istanzia in primis un’ ArrayList di Stub se schedule è nullo, altrimenti svuota il suddetto.

Istanzia un XmlPullParserFactory e da questa crea un parser, tramite il parser legge tag del file xml e le distingue in START\_TAG, TEXT, END\_TAG ed END\_DOCUMENT.

Finché non incontra tag di tipo END\_DOCUMENT in corrispondenza di START\_TAG crea un’istanza di Stub se il titolo della tag è "course", in corris-pondenza di TEXT copia il testo nel tag e infine in corrispondenza di END\_TAG copia il titolo della tag tramite parser.getName, in base al quale definisce i parametri dell’istanza di Stub appena creata.

case XmlPullParser.*END\_TAG*:  
 if (tagname.equalsIgnoreCase("course")){  
 this.getSchedule().add(stub);  
  
 } else if (tagname.

equalsIgnoreCase("teacher")) {  
 stub.setTeacherName(text);  
  
 } else if (tagname.

equalsIgnoreCase("name")) {  
 stub.setCourseName(text);  
  
 } else if (tagname.

equalsIgnoreCase("day")) {  
 stub.setDay(Integer.*parseInt*(text));  
 }  
 break;  
  
 default:  
 break;  
 }  
 eventType = parser.next();  
 }  
  
 } catch (XmlPullParserException | IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public Schedule getDailySchedule(int day){  
 Schedule schedule = new Schedule(new ArrayList<Stub>());  
 for (Stub stub : this.schedule.getSchedule()){  
 if (stub.getDay() == day)  
 schedule.getSchedule().add(stub);  
 }  
 return schedule;  
 }  
 }

Il metodo getDailySchedule restituisce un ArrayList di Schedule in base al Calendar.get-WeekDay datogli, scorrendo tutta la schedule e confrontando con il day immesso .

Stub

Module: MyLibrary

public class Stub implements Parcelable {  
 private int year, day;  
 private String courseName, teacherName, room;  
 private double startTime, endTime;  
  
 //Constructors  
 public Stub(int y, int d, String cN, String tN,   
 String r, double sT, double eT){  
 this.year = y;  
 this.day = d;  
 this.courseName = cN;  
 this.teacherName = tN;  
 this.room = r;  
 this.startTime = sT;  
 this.endTime = eT;  
 }  
  
 public Stub(){  
 //empty  
 }  
//Setters  
 public void setYear(int year) {  
 this.year = year;  
 }  
  
 public void setDay(int day) {  
 this.day = day;  
 }  
  
 public void setCourseName(String courseName) {  
 this.courseName = courseName;  
 }  
  
 public void setTeacherName(String teacherName) {  
 this.teacherName = teacherName;  
 }  
  
 public void setRoom(String room) {  
 this.room = room;  
 }  
  
 public void setStartTime(double startTime) {  
 this.startTime = startTime;  
 }  
  
 public void setEndTime(double endTime) {  
 this.endTime = endTime;  
 }  
  
 public int getYear() {  
 return year;  
 }  
  
 public int getDay() {  
 return day;  
 }

Stub è una classe semplice che contiene al suo interno valori base.

Verrà implementata l’interfaccia Parceable per renderlo trasmissibile tramite Intent.

public String getTeacherName() {  
 return teacherName;  
 }

public String getTeacherName() {  
 return teacherName;  
 }

public String getRoom() {  
 return room;  
 }  
  
 public double getStartTime() {  
 return startTime;  
 }  
  
 public double getEndTime() {  
 return endTime;  
 }  
  
 @Override  
 public int describeContents() {  
 //*TODO* return 0;  
 }  
  
 @Override  
 public void writeToParcel(Parcel dest, int flags) {  
 //*TODO* }  
}



Commento e Codice

**Modulo: Wear**

Wearable\_Activity\_Main

Module: Wear

public class Wearable\_Activity\_Main extends WearableActivity  
 implements Wearable\_Fragment\_Stub.  
 OnFragmentInteractionListener{  
  
  
 //*TODO Implementare swipe per passare tra le attività* //*TODO passare tramite intento l'aula che si vuole   
 visualizzare sulla mappa* //*Todo passare tramite intent il professore di cui si vuole   
 visualizzare il profilo* //*todo trovare un modo per accedere a tutti i profili dei   
 professori e listarne almeno le mail* private ArrayList<Stub> stubList;  
 private int tempStub;  
 String path = "/my\_path";  
 String TAG = "Wearable\_device";  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 stubList = new ArrayList<>();  
 setStubList();  
 setContentView(R.layout.*wearable\_activity\_main*);  
  
 //Ricevitore di messaggi  
 //discerne tra i messaggi che arrivano al messageService quelli che gli interessano  
 LocalBroadcastManager.

*getInstance*(this).registerReceiver(  
 new Receiver(),  
 new IntentFilter(Intent.*ACTION\_SEND*)  
 );  
 }  
  
  
  
 private void setStubList() {  
 stubList.clear();  
 stubList.add(new Stub(1,1,"1","1","1",1.1,1.1));  
 stubList.add(new Stub(2,2,"2","2","2",2.2,2.2));  
 }  
  
 //BUTTONS  
  
 public void nextStubIntent(View v){  
  
 }  
  
 private Integer getCurrentStubIndex(ArrayList<Stub> stublist, double cTime){  
 for(int i=0; i<stublist.size();i++){  
 if (cTime > stubList.get(i).getStartTime() && cTime < stubList.get(i).getEndTime())  
 return i;  
 }  
 return null;  
 }

In generale le Activity del modulo Wear sono ancora una bozza, infatti sono presenti numerose pezze e parti sostituibili con metodi già presenti nel modulo Mobile (ad esempio non ci sono oggetti Schedule ma solo ArrayList di Stub).

Alla creazione della Wearable\_Activity\_Main si aggiornano i valori della stubList e allo stesso modo del modulo mobile si filtrano i messaggi ricevuti tramite IntentFilter.

Il metodo setStubList è attualmente atto al testing, in quanto attribuisce dei valori non reali per costruire due Stub di prova.

nextStubIntent è l’OnClickListener del pulsante ‘Next’, dovrebbe far passare il ViewPager al prossimo Wearable\_Fragment\_Stub.

getCurrentStubIntex verrà spostato alla classe Stub, restituisce l’indice dello Stub corrispon-dente all’ora attuale datagli una dailySchedule .

//NOTIFICHE  
 private NotificationChannel   
 createChannel(NotificationManager nManager){  
 String id ="my\_channel\_01";  
 CharSequence name = "channel-name";  
 int importance = nManager.*IMPORTANCE\_DEFAULT*;  
  
 NotificationChannel nChannel = new   
 NotificationChannel(id, name,importance);  
  
 String description = "description";  
 nChannel.setDescription(description);  
 nChannel.enableLights(true);  
 nChannel.enableVibration(true);  
  
 nManager.createNotificationChannel(nChannel);  
  
 return nChannel;  
 }  
 public void getNotified(View view){  
 NotificationManager mNotificationManager = (  
 NotificationManager)   
 getSystemService(Context.*NOTIFICATION\_SERVICE*);  
  
  
 Intent intent = new Intent(this,   
 AttendanceService.class );  
 PendingIntent pi = PendingIntent.   
 *getActivity*(this, 0, intent, 0);  
  
 NotificationCompat.Builder builder =   
 new NotificationCompat  
 .Builder(this, createChannel(mNotificationManager)  
 .getId());  
 builder.setContentTitle("titolo della notifica");  
 builder.setContentText("contenuto della notifica");  
 builder.setSmallIcon(  
 R.drawable.*common\_google\_signin\_btn\_icon\_dark*);  
 builder.addAction(R.drawable.*ic\_check*, "Will\_attend", pi);  
  
 Notification notification = builder.build();  
  
 NotificationManagerCompat nmc =   
 NotificationManagerCompat.*from*(this);  
 nmc.notify(1,notification);  
 }  
  
 //COMMUNICATION  
 public void sendCommunication(){  
 new MessageSenderThread(path, "I just sent the handheld a message").start();  
 }  
  
  
 public class Receiver extends BroadcastReceiver{  
 @Override  
 public void onReceive(Context context, Intent intent) {  
 Log.*v*(TAG, "just received a message");  
 }  
 }

Il metodo createChannel crea appunto un canale (definendone id,name, descrizione, importance e se deve o meno attivare luci e vibrazione) tramite il cui l’applicazione può gestire le proprie notifiche. Necessario per il NotificationCompat-.Builder.

Il metodo getNotified forza una notifica tramite il builder, la notifica contiene anche un pulsante tramite il quale l’utente può attestare la propria presenza alla prossima lezione. Questa funzione non è implementata.   
Il pulsante è definito tramite setSmallIcon, e la rispettiva azione tramite addAction

SendCommunication invia un messaggio di test al dispositivo mobile.

Receiver notifica tramite terminale se si è ricevuta un messaggio dal dispositivo mobile.

class MessageThread extends Thread{  
 String path;  
 String message;

MessageThread(String p, String m){  
 path = p;  
 message = m;  
 }  
  
 public void run(){  
 Task<List<Node>> wearableList =   
 Wearable.*getNodeClient*(getApplicationContext())

.getConnectedNodes();  
 try{  
 List<Node> nodes = Tasks.*await*(wearableList);  
 for (Node node : nodes){  
 Task<Integer> sendMessageTask =  
   
 Wearable.*getMessageClient*(Phone\_Activity\_Main.this)  
 .sendMessage(  
 node.getId(), path, message.getBytes());  
  
 try{  
 Tasks.*await*(sendMessageTask);  
 messageText(  
 "I sent a message to the wearable");  
 } catch (  
 ExecutionException | InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 } catch (

InterruptedException | ExecutionException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 //OTHERS  
 private void setTxtView(int campotisesto, String valore){  
 TextView tw;  
 tw = findViewById(campotisesto);  
 tw . setText(valore);  
 }  
  
 @Override  
 public boolean onTouchEvent(MotionEvent event) {  
  
 int action = MotionEventCompat.*getActionMasked*(event);  
 String DEBUG\_TAG = "testing\_wearable";  
 switch(action) {  
 case (MotionEvent.*ACTION\_DOWN*):  
 Log.*d*(DEBUG\_TAG, "Action was DOWN");  
 return true;  
 case (MotionEvent.*ACTION\_MOVE*):  
 Log.*d*(DEBUG\_TAG, "Action was MOVE");  
 return true;  
 case (MotionEvent.*ACTION\_UP*):  
 Log.*d*(DEBUG\_TAG, "Action was UP");  
 return true;  
 case (MotionEvent.*ACTION\_CANCEL*):  
 Log.*d*(DEBUG\_TAG, "Action was CANCEL");  
 return true;  
 case (MotionEvent.*ACTION\_OUTSIDE*):  
 Log.*d*(DEBUG\_TAG, "Movement occurred outside bounds " +  
 "of current screen element");  
 return true;

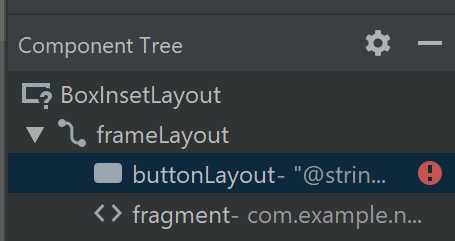
Questa classe MessageThread è stata già implementata nel modulo Mobile, e dev’essere infatti spostata in MyLibrary.

setTxtView ha la funzione di rendere più leggibile il codice.

onTouchEvent è un abbozzo di implemen-tazione delle gesture touch; sarà sostituito da ViewPager in quanto queste sono già implementate. Verranno richieste solo le gesture di scorrimento in basso e in alto

default:  
 return super.onTouchEvent(event);  
 }  
  
 }  
 @Override  
 public void onFragmentInteraction(Uri uri) {  
 ImageButton mapButton = findViewById(R.id.*mapImagebutton*);  
 ImageButton teacherButton = findViewById(R.id.*teacherImageButton*);  
  
 mapButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent (getApplicationContext(), Wearable\_Activity\_Map.class);  
 startActivity(intent);  
 }  
 });  
  
 teacherButton.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
  
 }  
 });  
 }  
}

onFragmentInteration è un metodo in testing per creare gli OnClickListener dei pulsanti mapImageButton e teacherImageButton nel   
frammento, ma è tuttora inutilizzabile. Entrambi dovrebbero chiamare un Intent atto a creare le rispettive Activity

L’errore di buttonLayout indica che non corrisponde nessun metodo ‘getNext’, ma deve semplicemente essere cambiato in public anziché private

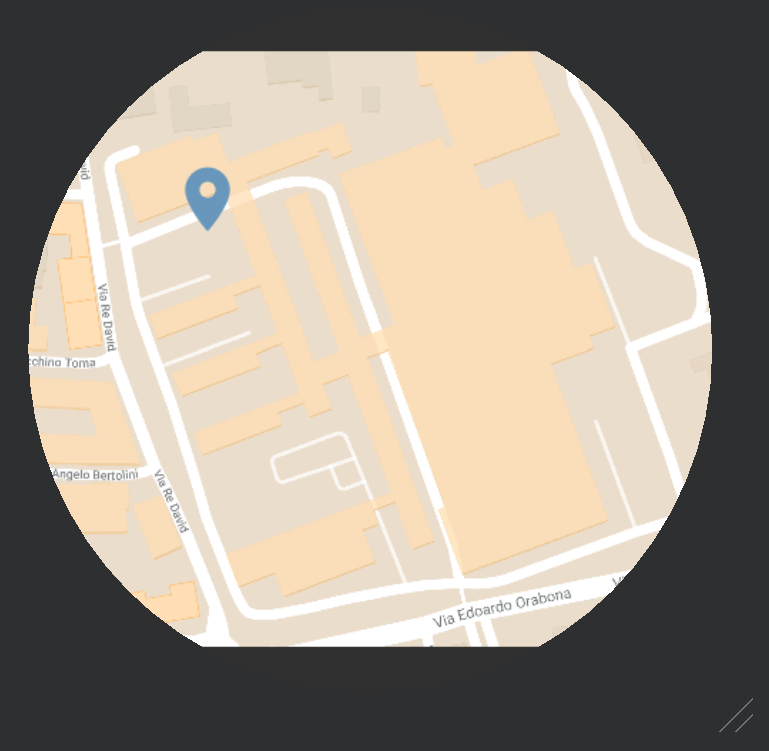
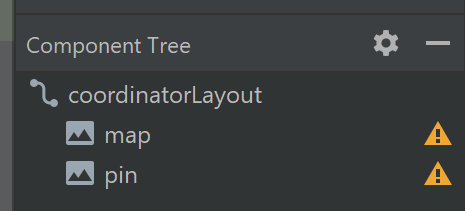
Wearable\_Activity\_Map

Module: Wear

public class Wearable\_Activity\_Map extends   
 AppCompatActivity {  
 final ImageView pin findViewById(R.id.*pin*);;  
 //*TODO bisogna salvare tutte le posizioni delle aule   
 in values.room\_coordinates.xml* @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 updatePinPosition();  
 setContentView(R.layout.*wearable\_activity\_map*);  
  
 }  
  
 private void updatePinPosition(){  
 //*todo get iv x and y values by   
 values.room\_coordinates.xml* }  
}

Il fine dell’activity Wearable\_Activity\_Map è di far visualizzare all’utente la mappa del Poliba e posizionare un’icona pin nel punto corrispon-dente all’aula cercata. Per implementarla bi-sogna salvare individualmente la posizione di ciascuna aula in values.room\_coordinates.xml

updatePinPosition assegna I constraint dell’ImageView pin nel costraintLayout principale

Entrambe le ImageView non hanno constraint perché il pin verrà impostato a runtime, mentre map rimarrebbe comunque in posizione 0x0

Wearable\_Fragment\_Stub

Module: Wear

public class Wearable\_Fragment\_Stub extends Fragment {  
 private String teacher;  
 private String course;  
 private String classroom;  
 private String startTime;  
 private String endTime;  
  
 private OnFragmentInteractionListener mListener;  
  
 public Wearable\_Fragment\_Stub() {  
 // Required empty public constructor  
 }  
  
 public static Wearable\_Fragment\_Stub newInstance(Stub stub)   
 {  
 Wearable\_Fragment\_Stub fragment =  
 new Wearable\_Fragment\_Stub();  
 Bundle args = new Bundle();  
  
 args.putString("teacher", stub.getTeacherName());  
 args.putString("course", stub.getCourseName());  
 args.putString("classroom", stub.getRoom());  
 args.putString("start time",   
 String.*valueOf*(stub.getStartTime()));  
 args.putString("end time",   
 String.*valueOf*(stub.getEndTime()));  
  
 fragment.setArguments(args);  
 return fragment;  
 }  
  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 if (getArguments() != null) {  
 teacher = getArguments().getString("teacher");  
 course = getArguments().getString("course");  
 classroom = getArguments().getString("classroom");  
 startTime = getArguments().getString("start time");  
 endTime = getArguments().getString("end time");  
 }  
 }  
  
 @Override  
 public View onCreateView(  
 LayoutInflater inflater,   
 ViewGroup container,  
 Bundle savedInstanceState) {  
 // Inflate the layout for this fragment  
 return inflater.inflate(  
 R.layout.*wearable\_fragment\_stub*,  
 container, false);  
 }

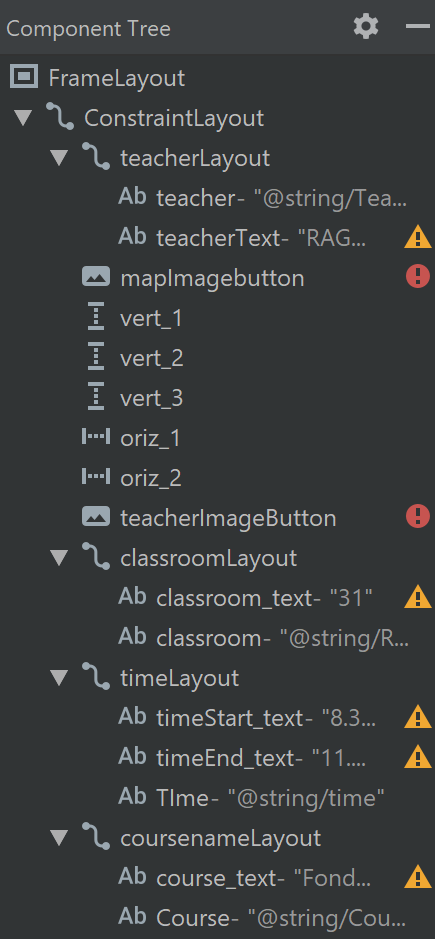
Wearable\_Fragment\_Stub è l’implementazione base del Fragment principale dell’Activity Wearable\_Activity\_Main.

La creazione di una nuova istanza istanzia i valori contenuti nello stub in input in un Bundle args, che vengono a loro volta inglobati negli Arguments del fragment tramite setArguments.

Alla creazione del fragment i valori da disporre vengono assegnati

onCreateView, onButtonPressed, onAttach e onDetach sono tutti metodi automaticamente implementati da Android Studio

public void onButtonPressed(Uri uri) {  
 if (mListener != null) {  
 mListener.onFragmentInteraction(uri);  
 }  
 }  
  
 @Override  
 public void onAttach(Context context) {  
 super.onAttach(context);  
 if (context instanceof OnFragmentInteractionListener) {  
 mListener = (OnFragmentInteractionListener) context;  
 } else {  
 throw new RuntimeException(context.toString()  
 + " must implement OnFragmentInteractionListener");  
 }  
 }  
  
 @Override  
 public void onDetach() {  
 super.onDetach();  
 mListener = null;  
 }

Gli errori riportati sono per HardCodedText di prova di visualizzazione

C’è bisogno di migliorare la leggibilitàWearable\_Activity\_Teacher

Module: Wear

L’activity Wearable\_Activity\_Teacher dev’essere ancora inizializzata