Interaktion I - JOGGA

JOGGA

Der Jogging-Companion





T.Höfer | S.Berthold | IAM 6

GLIEDERUNG:

(1) Idee

(2) Funktionen

(3) Codebeispiele



IDEE:

Jeder Jogger kennt das Problem: Man rafft sich auf um eine Runde zu laufen, schwitzt während dessen ohne Ende und vergisst dabei Raum und Zeit. Genau hier setzt JOGGA an: Man lädt das Programm einfach auf sein Android-Smartphone, nimmt es in der Hosentasche mit und ist während des Laufens sowie nach Beendingung über alle Statistiken bestens informiert!





Projektüberblick

- Programmierung in Java
- Android Dalvik VM != SUN JVM

- GPS-Sensor → Strecke
- Beschleunigungsmesser → Schrittzähler
- MediaPlayer
- Joggingstatistiken twittern
- Graph zur Geschwindigkeit & min/km (= Bewertungseinheit)







T.Höfer | S.Berthold | IAM 6

Views

=> Komponenten werden ähnlich Flex über XML zusammengebaut

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@drawable/arb_start_background">

    <TextView
     android:id="@+id/joggingDuration"
     android:layout_marginTop="10dip"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:textStyle="bold"
     android:gravity="center_horizontal"
     android:text="00:21:55"
     android:layout_alignParentBottom="false"
     android:shadowDy="-1.0"
     android:shadowDx="-1.0"
     android:shadowRadius="0.1"
     android:shadowColor="#ffffff"
     android:textSize="65dip"
     android:textColor="#ff8226"/>

</RelativeLayout>
```



FUNKTIONEN/CODEBEISPIEL – Strecke & Geschwindigkeit

- Berechnungen gekapselt im GpsModel
- Aktualisiert sich bei jedem Update der GPS-Position

```
public void addLocation(Location location, Long time) {
    if(lastLocation != null) {
        float deltaDistance = lastLocation.distanceTo(location);
        long deltaTime = time - lastTime;

        // Enthält die aktuelle Distanz
        currentDistance += deltaDistance;

        // Aktuelle Geschwindigkeit berechnen in m/s
        JoggingModel.setSpeed(deltaDistance / deltaTime * 1000);
    }
    getData().add(location);
    lastLocation = location;
    lastTime = time;
}
```



FUNKTIONEN/CODEBEISPIEL – Mediaplayer

° Ein Verzeichnis wird rekursiv nach .mp3-Dateien durchsucht

° Jede relevante Datei wird als Song-Instanz gespeichert

```
public class Jukebox {
    …
    // Liefert per Zufall einen Song aus der Liste
    public Song nextSong() {
        int index = (int)(Math.random() * songList.size()-1);
        return songList.get(index);
    }
    …
}
```

```
public class Song {

    protected String filename;
    protected String path;

    … getter & setter ...

}
```



FUNKTIONEN/CODEBEISPIEL – Schrittzähler

° Wie werden Schritte ermittelt?

```
float x = event.values[0];
float y = event.values[1];
float z = event.values[2];
// Länge Beschleunigungsvektor berechnen
float val = (float)Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2) + Math.pow(z, 2));
// Gravitation abziehen
val -= SensorManager.STANDARD_GRAVITY;

int result = (int)(val * 100);
// Positiv => Schrittanfang
if(result >= 50) {
    state = DIR.TOP;
// Negativ => Schritt vollendet
} else if(result <= -50 && state != null) {
    if(state == DIR.TOP) {
        steps++;
        setChanged();
        notifyObservers(steps);
    }
    state = DIR.BOTTOM;
}
```



FUNKTIONEN/CODEBEISPIEL – Schrittzähler

° Nutzung des Accelerometers (= Klasse Pedometer)

° Implementierung mit Observer-Pattern

```
// Pedometer von Observable ableiten
// => erbt Methoden notifyObservers() und addObserver()
public class Pedometer extends Observable {
    notifyObservers(anzahlSchritte);
}


Pedometer pedometer = new Pedometer();
pedometer.addObserver(new Observer() {
    // Listener-Methode update() bei jedem Schritt aufrufen
    @Override
    public void update(Observable observable, Object data) {
        joggingSteps.setText(data + " Schritte");
    }
});
```



FUNKTIONEN/CODEBEISPIEL – Twittern

° Verwendung eines „öffentlichen Intent" des Programms Twidroid

° Entspricht Verwendung einer Drittkomponente

° Implementierung trivial:

```
Intent intent = new Intent("com.twidroid.SendTweet");
intent.putExtra("com.twidroid.extra.MESSAGE", "Mal wieder gelaufen mit JOGGA! Dauer: " +
FormModel.getDurationString() + " Min., Strecke: " + FormModel.getDistanceString() + ", Min/Km: " +
FormModel.getMinPerKm());
intent.setType("application/twitter");
try {
     startActivity(intent);
} catch (ActivityNotFoundException e) {
     Toast.makeText(getApplicationContext(), "Twidroid nicht installiert!", Toast.LENGTH_SHORT).show();
}
```

° **Wir haben das Rad nicht neu erfunden! :)**



FUNKTIONEN/CODEBEISPIEL – Graphdarstellung

° Beliebige Messreihen können dargestellt werden: Interface IPlottable

```
public interface IPlottable {
    public void addData(Float f);
    public List<Float> getData();
    public Float getMin();
    public Float getMax();
    public int getColor();
    public String display(float val);
}
```

° Ständiges Rendern & Löschen eines PNG in separatem Thread



Interaktion I - JOGGA

```
public class PerformanceMesser implements IPlottable {

	protected List<Float> data = new Vector<Float>();

	public void addData(Float f) {
		data.add(f);
	}
	public List<Float> getData() {
		float average = 0;
		List<Float> vals = new LinkedList<Float>();

		// Graph soll später nur aus 5 Abschnitten bestehen
		int div = (int)(data.size() / 5);
		if(div == 0)
			return null;
		for(int i = 0; i < data.size(); i++) {
			float v = data.get(i).floatValue();
			average += v;
			// Alle Anzahl div Elemente Durchschnitt berechnen
			if(i % div == 0) {
				vals.add(average / div);
				average = 0;
			}
		}
		return vals;
	}

	public Float getMax() {
		return Collections.max(data);
	}

	public Float getMin() {
		return Collections.min(data);
	}

	public int getColor() {
		return Color.argb(0xff, 0xff, 0x66, 0);
	}

}
```



FUNKTIONEN/CODEBEISPIEL – Threading

° Haupt-Thread => Touch-Events

° Drawer-Thread => Generiert den Graphen ständig neu

° Loop-Thread => Thread, der die Zahlenwerte für das UI berechnet bzw. Methoden dafür aufruft

Probleme & Lösung

° Manche Daten werden von mehreren Threads gleichzeitig genutzt

° Verwendung von Vector als Klasse für Listen => Zugriff threadsicher

° Bereiche in denen ein Thread exklusiven Zugriff auf eine Datenstruktur benötigt mit „synchronized" gekennzeichnet.



Vielen Dank für die Aufmerksamkeit!

