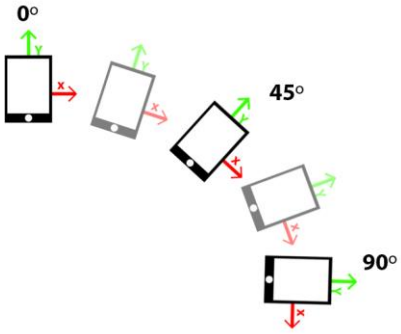
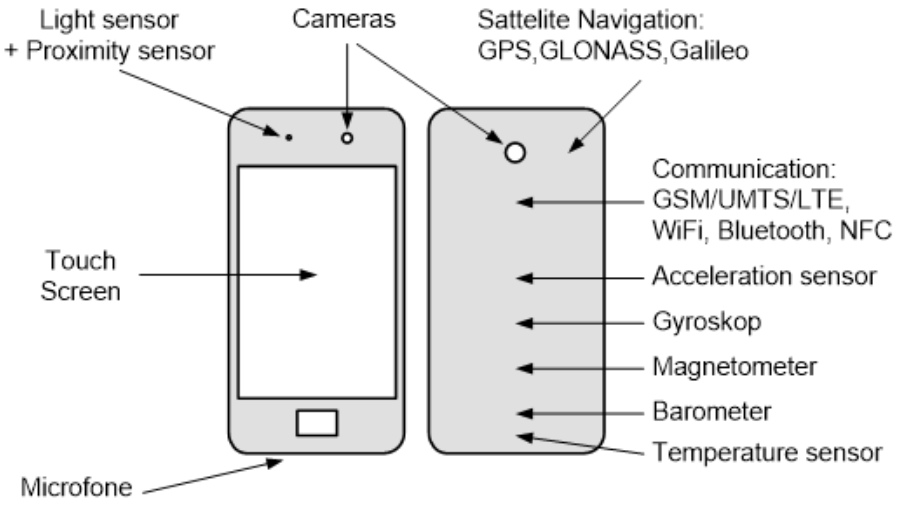
TEMAT: Sprzętowe zasoby systemu.

WPROWADZENIE

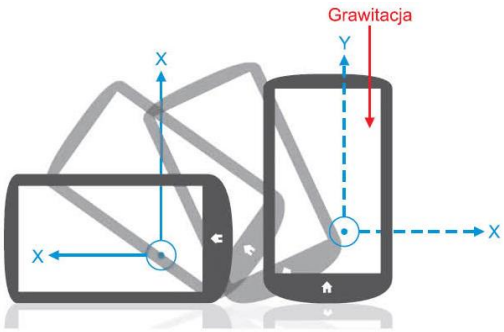
1. **Sensory** są zasobami sprzętowymi smartfonów. Ich dostępność umożliwia tworzenie przydatnych aplikacji wspomagających użytkownika w różnych sytuacjach (odnalezienie lokalizacji, pomiar temperatury, odblokowanie telefonu poprzez odczytanie linii papilarnych lub rozpoznanie twarzy), bądź dostarczających ciekawej rozrywki (gry).
2. Urządzenia z systemem Android wyposażone są w rozmaite sensory, dokonujące różnorakich pomiarów. Można je podzielić na trzy główne kategorie:
    a. Czujniki ruchu (przyspieszenie, obrót),
    b. Czujniki środowiskowe (ciśnienie, temperatura, wilgotność, natężenie światła),
    c. Czujniki położenia (fizyczna lokalizacja urządzenia – np. magnetometr, czujniki orientacji).
3. Inny podział sensorów:
    a. Sprzętowe (hardware-based) – fizyczne komponenty wbudowane w urządzenie,
    b. Aplikacyjne (software-based) – wirtualne czujniki, które wykorzystują dane dostarczane z jednego lub kilku sensorów sprzętowych.



4. Należy pamiętać, że każdy sprzęt wyposażony jest w zestaw różnych czujników (występowanie sensorów jest zależne od konkretnego modelu). **Uwaga:** zawsze dbaj o to, aby upewnić się, czy użytkownik pobierający Twoją aplikację posiada w swoim urządzeniu wymagane sensory. Jeśli działanie aplikacji opiera się na wykorzystaniu czujników (czyli w przypadku ich braku nie będzie dostępna główna funkcjonalność) należy uniemożliwić użytkownikowi pobranie aplikacji, jeśli nie posiada niezbędnych sensorów. Można to zrobić z poziomu pliku *AndroidManifest.xml*.
5. Przykład ulokowania sensorów:



6. Wykorzystanie czujników z poziomu aplikacji: *Sensors Framework*.
7. Przykłady sensorów:
    a. Akcelerometr (`TYPE_ACCELEROMETER`) – służy do pomiaru przyspieszeń liniowych lub kątowych w trzech wymiarach (X, Y, Z) . Odpowiada za detekcję ruchu. Umożliwia automatyczne wykrywanie ułożenia przestrzennego urządzenia (pochylenie, przekręcenie) oraz sterowanie jego funkcjami przez poruszanie nim. Gdy osie x lub y ekranu urządzenia pokrywają się z wektorem grawitacji, możliwe jest określenie orientacji wyświetlacza.

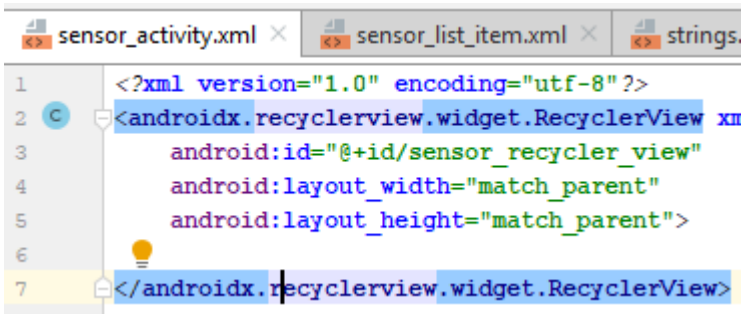


    Pomiar wartości:
    - SensorEvent.values[0] – oś X
    - SensorEvent.values[1] – oś Y
    - SensorEvent.values[2] – oś Z

    Uwaga: siła grawitacji ma zawsze wpływ na wynik. W przypadku urządzenia leżącego na stole zmierzona wartość przyspieszenia będzie wynosiła $9{,}81\frac{m}{s^2}$ (przyspieszenie ziemskie). W przypadku swobodnego upadku wartość ta będzie bliska zeru.
    b. Żyroskop (`TYPE_GYROSCOPE`) – wykorzystywany do pomiaru położenia kątowego. Mierzy prędkość kątową obracających się obiektów wokół osi X, Y lub Z. Służy do precyzyjnego określania orientacji smartfona.
    c. Czujnik zbliżeniowy (`TYPE_PROXIMITY`) – reaguje na zbliżanie się obiektu. Zwykle raportuje odległość w cm. Zdarza się, że rejestruje jedynie dwie opcje: blisko, daleko. Czujnik odnotowuje w takim wypadku zbliżenie, gdy obiekt przekroczy pewien próg. Zwykle wykorzystywany do sprawdzania, czy użytkownik rozmawia przez telefon (ma przyłożony do ucha smartfon), aby wygasić ekran.
    d. Czujnik kroków (`TYPE_STEP_DETECTOR`) – odnotowuje zdarzenie, gdy użytkownik zrobi krok. Zwracane wartości to 0 i 1.
    e. Czujnik światła (`TYPE_LIGHT`) – reaguje na zmianę intensywności docierającego do niego strumienia światła (jednostka: luks, lx).
    f. Czujnik ciśnienia (`TYPE_PRESSURE`) – mierzy ciśnienie powietrza (jednostka: hPa lub mbar).
    g. Czujnik wilgotności (`TYPE_RELATIVE_HUMIDITY`) – dokonuje pomiaru względnej wilgotności (w procentach).
8. Czujniki ruchu i położenia wymagają zwykle kalibracji. Czujniki środowiskowe są łatwiejsze w użyciu – przeważnie nie są konieczne dodatkowe ustawienia.
9. Urządzenie może być wyposażone w kilka czujników tego samego typu.

TREŚĆ ZADANIA

1. **Cel:** utworzenie aplikacji *SensorApp* do zapoznania się z zasobami sprzętowymi urządzeń z systemem Android, a w szczególności:
    a. zbadanie, jakie czujniki są dostępne w posiadanym urządzeniu lub wybranym emulatorze;
    b. wyświetlenie informacji o sumie wszystkich dostępnych czujników;
    c. wyświetlanie wartości wybranych dwóch czujników po kliknięciu w odpowiedni element listy;
    d. pobranie i geokodowanie lokalizacji.
2. Utwórz aplikację o nazwie *SensorApp*, wybierając główną aktywność typu *Empty Activity*. Głównej aktywności nadaj nazwę *SensorActivity*, zaś powiązany plik układu widoku nazwij *sensor_activity.xml*.
3. W pliku układu *sensor_activity* dodaj listę typu RecyclerView .



Stwórz plik układu do zdefiniowania układu dla pojedynczego elementu listy: *sensor_list_item.xml* tak, aby umożliwić wyświetlanie na liście ikony wskazującej, że jest to czujnik oraz nazwy czujnika. Zadbaj o poprawne ustawienia odstępów oraz zapewnienie skalowalności obrazka (*wiedza z poprzedniego zadania*).

4. Pobierz listę wszystkich sensorów urządzenia w *SensorActivity* w metodzie *onCreate()*:

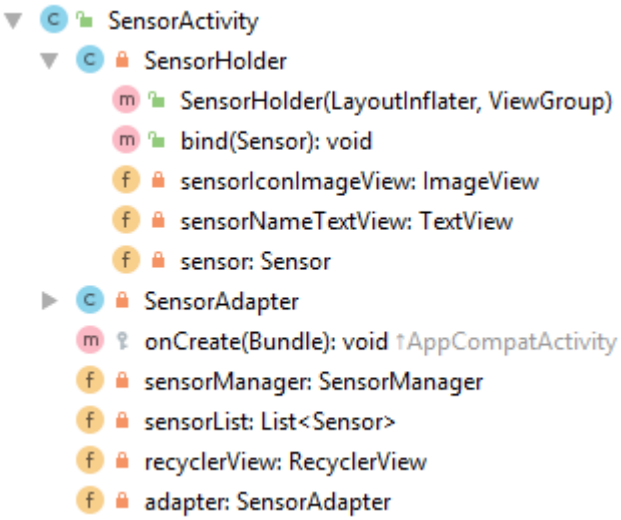
```
private SensorManager sensorManager;
private List<Sensor> sensorList;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.sensor_activity);

    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);
}
```

5. Dodaj wewnętrzne klasy adaptera oraz holdera i zapewnij wyświetlanie listy sensorów w widoku RecyclerView.

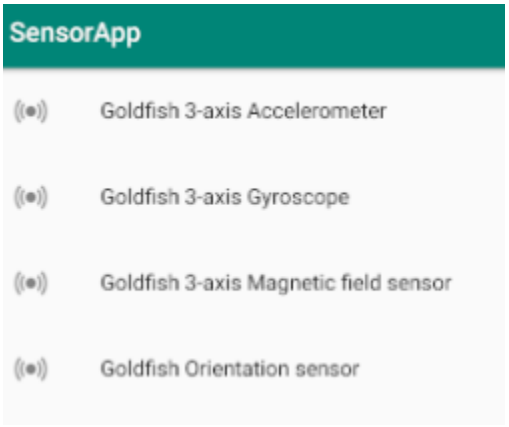
6. Zaktualizuj kod metody *onCreate()* tak, aby *RecyclerView* korzystał z pomocniczych klas (podłącz *SensorAdapter*):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.sensor_activity);

    recyclerView = findViewById(R.id.sensor_recycler_view);
    recyclerView.setLayoutManager(new LinearLayoutManager( context: this));

    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);

    if (adapter == null) {
        adapter = new SensorAdapter(sensorList);
        recyclerView.setAdapter(adapter);
    } else {
        adapter.notifyDataSetChanged();
    }
}
```

7. Struktura kodu klasy aktywności *SensorActivity*:



8. Sprawdź, czy czujniki wyświetlają się poprawnie na liście w głównej aktywności. Przykładowe uruchomienie aplikacji na emulatorze:



9. Po dłuższym przyciśnięciu (long click) nazwy każdego z czujników na liście powinno pojawiać się okno dialogowe typu AlertDialog wyświetlające szczegóły dotyczące czujnika (np. informacje o producencie (*sensor.getVendor()*) oraz maksymalnej zwracanej wartości (*sensor.getMaximumRange()*) → https://developer.android.com/develop/ui/views/components/dialogs

10. Podobnie jak w poprzednim zadaniu (nr 4), dodaj w menu przycisk, który będzie odpowiadał za dodawanie do paska narzędzi napisu z liczbą wszystkich czujników.

```
<string name="sensors_count">Liczba czujników: %d</string>
```

11. Z listy wyświetlanych sensorów wybierz dwa, dla których po kliknięciu będzie można podejrzeć szczegóły, dotyczące zarejestrowanych wartości, zmierzonych przez dany czujnik. Wyróżnij je na liście. Dalsza część opisu będzie zawierała instrukcję dodania pojedynczego czujnika, **jednak należy analogicznie zaimplementować możliwość wyświetlenia wartości drugiego wybranego sensora. Uwaga: obsługa obydwu czujników powinna odbywać się w ramach jednej aktywności.**

12. Dodaj nową aktywność typu *Empty Activity*. Nazwij ją *SensorDetailsActivity.*

13. Przygotuj widok do wyświetlania wartości sensora. Wyświetl również informację o tym, jaki to sensor. W razie braku danego sensora na urządzeniu, należy przygotować następujący komunikat:

```
<string name="missing_sensor">Brak czujnika</string>
```

14. W utworzonej klasie *SensorDetailsActivity* dodaj następujące pola:

```
private SensorManager sensorManager;
private Sensor sensorLight;
private TextView sensorLightTextView;
```

15. W metodzie *onCreate()* klasy *SensorDetailsActivity* pobierz sensor wybranego typu, np.:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sensor_details);

    sensorLightTextView = findViewById(R.id.sensor_light_label);

    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sensorLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);

    if (sensorLight == null) {
        sensorLightTextView.setText(R.string.missing_sensor);
    }
}
```

16. Koniecznie upewnij się, czy dany sensor jest dostępny w urządzeniu (sprawdzenie, czy nie jest nullem) – jeśli go nie ma, wyświetl przygotowaną informację o braku czujnika.
17. Dodaj implementację interfejsu *SensorEventListener:*

```
public class SensorDetailsActivity extends AppCompatActivity implements SensorEventListener {
```

18. Zaimplementuj metodę *onStart():*

```
@Override
protected void onStart() {
    super.onStart();

    if (sensorLight != null) {
        sensorManager.registerListener( listener: this, sensorLight, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
```

Dlaczego rejestracja działania sensora nie powinna odbywać się w metodzie *onCreate*?

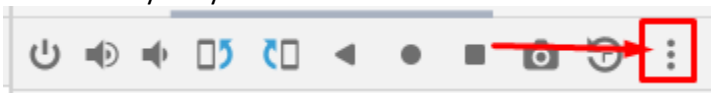
19. Zadbaj o wstrzymanie działania czujnika, gdy aplikacja nie będzie użytkowana. W przeciwnym wypadku zużycie baterii może drastycznie wzrosnąć. W tym celu zaimplementuj metodę *onStop():*

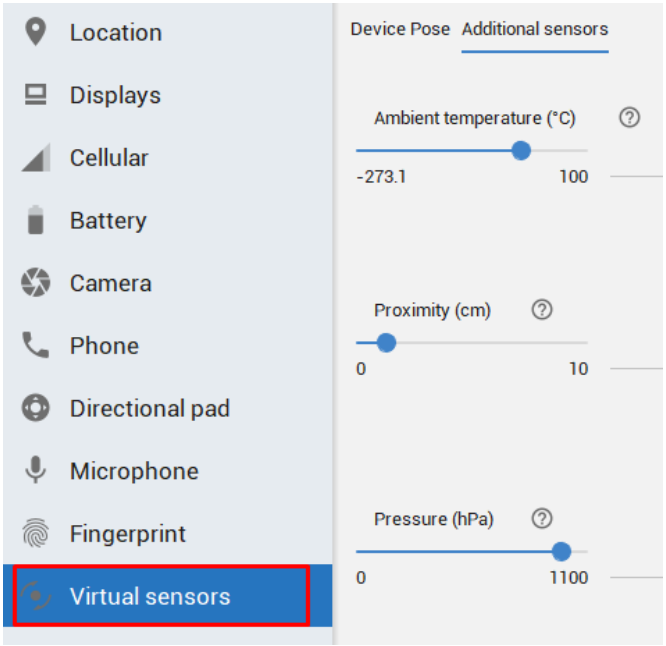
```
@Override
protected void onStop() {
    super.onStop();

    sensorManager.unregisterListener(this);
}
```

20. Zaimplementuj metodę *onSensorChanged()* – zadbaj o to, aby wyświetlały się w niej wartości odpowiedniego czujnika:

```java
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    int sensorType = sensorEvent.sensor.getType();
    float currentValue = sensorEvent.values[0];

    switch (sensorType) {
        case Sensor.TYPE_LIGHT:
            sensorLightTextView.setText(getResources().getString(R.string.light_sensor_label, currentValue));
            break;
        default:
    }
}
```

21. Przeanalizuj zmiany wartości danych rejestrowanych przez wybrane czujniki. Jest to możliwe również z wykorzystaniem emulatora:





22. Ile wartości zwracanych jest przez wybrane przez Ciebie czujniki (*tablica values obiektu SensorEvent*)? Sprawdź, czy metoda *onAccuracyChanged* jest kiedykolwiek wywoływana.
23. Pamiętaj o wiedzy zdobytej na poprzednich zajęciach – wszystkie stałe związane z zasobami powinny być wydzielone do odpowiednich plików.
24. **Przygotuj projekt do użycia lokalizacji**. W oknie SDK Manager w zakładce SDK Tools zainstaluj *Google Play services*.
25. W pliku *build.gradle* dodaj następujący wpis:
    `implementation 'com.google.android.gms:play-services-location:X.X.X'`
    Zamień X na aktualną wersję.
26. Dodaj nową pustą aktywność o nazwie *LocationActivity*.
27. W edytorze widoku nowej aktywności umieść na dole przycisk z napisem „*Pobierz lokalizację*”.
28. Dodaj również pole tekstowe typu TextView z napisem „*Wciśnij przycisk, aby pobrać ostatnią lokalizację*”.
29. Dodaj kod uruchamiający nową aktywność po kliknięciu z listy czujników magnetometru.

30. Korzystanie z lokalizacji wymaga od użytkownika, aby przyznał odpowiednie uprawnienia. W pliku *AndroidManifest.xml* ustaw tuż nad znacznikiem *application* uprawnienie ACCESS_COARSE_LOCATION (wymagane) oraz ACCESS_FINE_LOCATION (dodatkowe, aby wyniki były jak najbardziej precyzyjne):

```
package="pl.edu.pb.sensorapp">

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

<application
```

31. Zaczynając od Androida w wersji 6.0 (API 23), nie wystarczy ustawić odpowiednie wpisy w pliku manifestu, aby poprawnie nadać uprawnienia. Dodatkowo wymagane jest poświadczanie szczególnie ważnych uprawień w trakcie działania aplikacji.
32. W klasie *LocationActivity* w metodzie *onCreate* zaimplementuj do pobranego przycisku OnClickListener oraz metodę *onClick* – wywołaj w niej nową metodę *getLocation*:

```
getLocationButton.setOnClickListener(v -> getLocation());

private void getLocation() {
    if (ContextCompat.checkSelfPermission( context: this,
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]
                        {Manifest.permission.ACCESS_FINE_LOCATION},
                REQUEST_LOCATION_PERMISSION);
    } else {
        Log.d(TAG,  msg: "getLocation: permissions granted");
    }
}
```

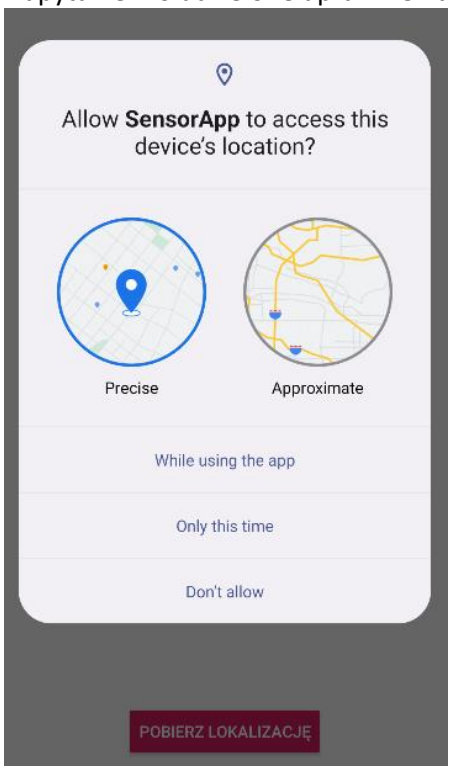
Metoda *getLocation* odpowiada za prośbę o pozwolenie typu ACCESS_FINE_LOCATION. Zdefiniuj stałe używane w metodzie:

- REQUEST_LOCATION_PERMISSION – stała typu int; służy do identyfikacji zgłoszenia uprawnienia; może to być każda liczba całkowita większa od 0.
- TAG – stała typu String; pozwala wskazać klasę, która odpowiada za utworzenie wiadomości zapisanej do logów.

33. Nadpisz metodę *onRequestPermissionsResult*:

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                       @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_LOCATION_PERMISSION:
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                getLocation();
            } else {
                Toast.makeText( context: this,
                        R.string.location_permission_denied,
                        Toast.LENGTH_SHORT).show();
            }
            break;
    }
}
```

34. Uruchom aplikację. Po kliknięciu przycisku „*Pobierz lokalizację*” w nowoutworzonej aktywności powinno pojawić się okno z zapytaniem o udzielone uprawnienia, jak poniżej:



Po przyznaniu uprawień lokalizacji kolejne kliknięcia przycisku „*Pobierz lokalizację*” nie będą miały widocznego efektu. Skoro uprawienia zostały przyznane, nie ma potrzeby kolejny raz o nie prosić. Powiadomienie pojawi się kolejny raz dopiero wtedy, gdy użytkownik odbierze uprawienie w ustawieniach aplikacji.

35. Następnie zaimplementuj właściwe pobieranie ostatniej lokalizacji. W tym celu na początku dodaj do pliku strings.xml następujący wpis:

```
<string name="location_text">"Szerokość geograficzna: %1$.4f \n Długość
geograficzna: %2$.4f \n Czas: %3$tr"</string>
```

36. W klasie *LocationActivity* dodaj trzy następujące zmienne:

```
private Location lastLocation;
private TextView locationTextView;
private FusedLocationProviderClient fusedLocationClient;
```

37. Dwie z nich powinny zostać zainicjalizowane w metodzie *onCreate*:

```
locationTextView = findViewById(R.id.textview_location);
fusedLocationClient = LocationServices.getFusedLocationProviderClient( activity: this);
```

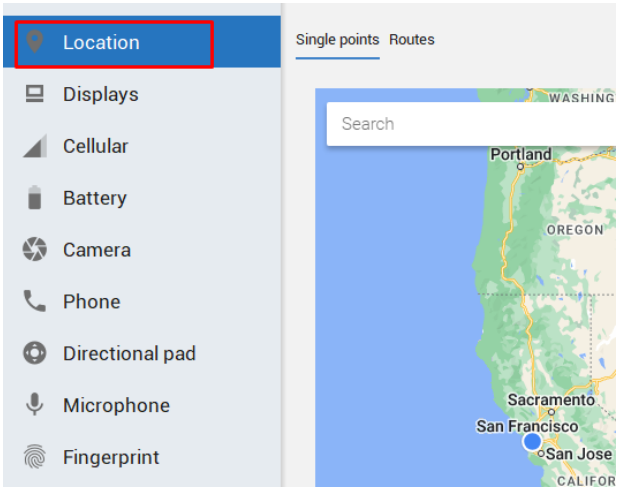
38. Dodaj następujący kod zamiast zapisywania komentarza do logów w metodzie *getLocation*:

```
fusedLocationClient.getLastLocation().addOnSuccessListener(
        location -> {
            if (location != null) {
                lastLocation = location;
                locationTextView.setText(
                        getString(R.string.location_text,
                                location.getLatitude(),
                                location.getLongitude(),
                                location.getTime()));
            } else {
                locationTextView.setText(R.string.no_location);
            }
        });
```

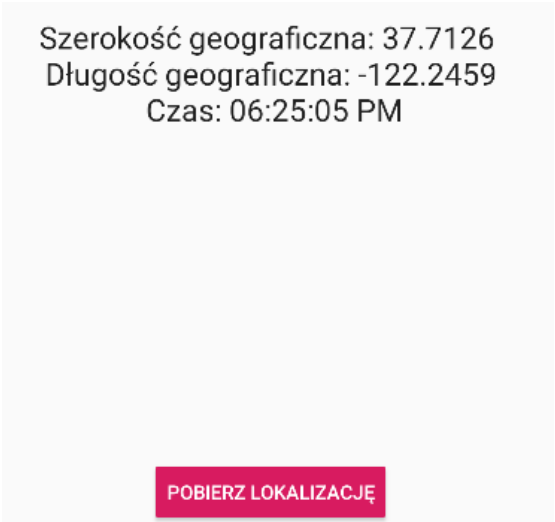
Utwórz również brakującą stałą tekstową o nazwie „no_location”.

39. Po poprawnym wykonaniu powyższych instrukcji powinna pojawić się możliwość wyświetlenia współrzędnych ostatnio pobranej lokalizacji.
WAŻNE: metoda *getLastLocation* pobiera jedynie ostatnią lokalizację, jaka jest zapisana w FusedLocationProviderClient. Jeśli żadna lokalizacja nie była pobierana od ostatniego restartu urządzenia, zwrócona zostanie wartość *null*.
40. Jeśli ostatnia lokalizacja nie wyświetla się (szczególnie w przypadku korzystania z emulatora), konieczne będzie jej zaktualizowanie. Najpierw wybierz dowolną lokalizację na mapie korzystając z opcji *Location* w rozszerzonych funkcjonalnościach emulatora.

Po tym uruchom aplikację Google Maps i wybierz przeliczanie trasy do dowolnego miejsca. Zaakceptuj wymagane uprawnienia. Aplikacja Google Maps powinna wykryć jako bieżącą lokalizację tą, która została ustawiona w emulatorze. W ten sposób można wymusić aktualizację lokalnej pamięci cache, z której lokalizację pobiera metoda *getLastLocation*.



41. Po wykonaniu opisanych kroków wróć do utworzonej aplikacji i kliknij przycisk „Pobierz lokalizację”. Powinny pojawić się wybrane współrzędne.



42. Jako że współrzędne nie są zazwyczaj przyjaznym użytkownikowi sposobem wyświetlania lokalizacji, w kolejnym kroku zadbamy o zamianę tych informacji na konkretny adres. Proces ten nosi nazwę geokodowania odwrotnego (ang. *reverse geocoding*).
43. Dodaj w widoku aktywności *LocationActivity* nowy przycisk (obok istniejącego) z napisem „*Pobierz adres*”. Dodaj również kolejne pole tekstowe, które będzie zawierało adres ostatniej lokalizacji.
44. Dodaj zmienną *addressTextView* typu *TextView.* W metodzie *onCreate* klasy *LocationActivity* zainicjalizuj nową zmienną:
```
addressTextView = findViewById(R.id.textview_address);
```
45. Dodaj następujący wpis do pliku *strings.xml:*
```
<string name="address_text">"Adres: %1$s \n Czas: %2$tr"</string>
```
46. W klasie *LocationActivity* dodaj nową metodę:

```
private String locationGeocoding(Context context, Location location) {
    Geocoder geocoder = new Geocoder(context, Locale.getDefault());
    List<Address> addresses = null;
    String resultMessage = "";

    try {
        addresses = geocoder.getFromLocation(
                location.getLatitude(),
                location.getLongitude(),
                maxResults: 1);
    } catch (IOException ioException) {
        resultMessage = context
                .getString(R.string.service_not_available);
        Log.e(TAG, resultMessage, ioException);
    }
```

Pierwsza część metody (na screenie powyżej) zawiera uruchomienie procesu geokodowania. W wyniku zwracana jest lista adresów. Ostatni parametr metody *getFromLocation* odpowiada za maksymalną liczbę adresów do odczytania. W bloku *catch* obsługiwane są ewentualne problemy z serwisem *Geocoder*.

47. Dodaj również dalszą część metody *locationGeocoding*:

```
    if (addresses == null || addresses.isEmpty()) {
        if (resultMessage.isEmpty()) {
            resultMessage = context
                    .getString(R.string.no_address_found);
            Log.e(TAG, resultMessage);
        }
    } else {
        Address address = addresses.get(0);
        List<String> addressParts = new ArrayList<>();

        for (int i = 0; i <= address.getMaxAddressLineIndex(); i++) {
            addressParts.add(address.getAddressLine(i));
        }
        resultMessage = TextUtils.join( delimiter: "\n", addressParts);
    }

    return resultMessage;
}
```

Druga część metody zawiera obsługę przypadku, w którym *Geocoder* nie będzie w stanie odnaleźć adresu dla podanych współrzędnych. Jeśli jednak uda się znaleźć adres, wszystkie jego linie zostaną złączone w jeden łańcuch znakowy. Wynikowy adres w postaci jednej linii tekstu jest zwracany w metodzie jako *resultMessage*.

48. Dodaj wywołanie utworzonej metody *locationGeocoding* w nowej metodzie *executeGeocoding*:

```
private void executeGeocoding() {
    if (lastLocation != null) {
        ExecutorService executor = Executors.newSingleThreadExecutor();
        Future<String> returnedAddress = executor.submit(() -> locationGeocoding(getApplicationContext(), lastLocation));
        try {
            String result = returnedAddress.get();
            addressTextView.setText(getString(R.string.address_text,
                    result, System.currentTimeMillis()));
        } catch (ExecutionException | InterruptedException e) {
            Log.e(TAG, e.getMessage(), e);
            Thread.currentThread().interrupt();
        }
    }
}
```

Metoda ta wywołuje geokodowanie w oddzielnym wątku (odpowiada za to interfejs *Executor*. Zwrócenie wartości wynikowej z wątku odbywa się przy użyciu interfejsu *Future*.

49. Dodaj wywołanie metody *executeGeocoding* jako wynik kliknięcia przycisku *„Pobierz adres”* – w tym celu dodaj następujący kod w metodzie *onCreate*:

```
Button getAddressButton = findViewById(R.id.button_address);
getAddressButton.setOnClickListener(v -> executeGeocoding());
```

Rezultat uruchomienia aktywności *LocationActivity*: