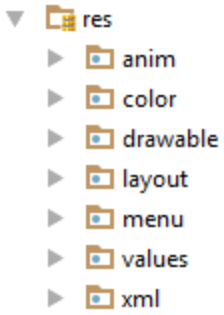
TEMAT: Programowe zasoby aplikacji.

WPROWADZENIE

1. Aplikacje dedykowane na platformę Android składają się z dwóch podstawowych elementów:
    a. Możliwości funkcjonalnych (instrukcji programu) - kod, który określa, w jaki sposób zachowuje się aplikacja,
    b. Zasobów – wszelkie dane, stałe wykorzystywane przez aplikację.
2. Przykładowe rodzaje zasobów:
    a. Łańcuchy znaków,
    b. Obrazy,
    c. Style i motywy,
    d. Definicje układu widoków,
    e. Animacje,
    f. Definicje elementów menu,
    g. Stałe liczbowe odpowiadające wartościom jednostek wielkości (wymiary),
    h. Pliki dźwiękowe,
    i. Konfiguracja funkcjonalności wyszukiwania.



3. Zasoby (czyli stała zawartość programu) zawsze powinny być wydzielane z właściwego kodu aplikacji, aby mogły być niezależnie utrzymywane.
4. Z każdym rodzajem zasobów mogą być związane domyślne oraz alternatywne pliki xml.
5. Zasoby należące do projektu przechowywane są w strukturze katalogów o ściśle określonej postaci. Wszystkie zasoby muszą być umieszone w katalogu res, w podkatalogach o konkretnych nazwach, zapisanych małymi literami. Nazwa każdego katalogu odpowiada rodzajowi zasobów, jakie są w nim przechowywane (jak na rysunku wyżej). Dodatkowo można tworzyć alternatywne (wyspecjalizowane) wersje zasobów, dbając o odpowiednią konwencję nazewniczą np.:

```
res/
    drawable/
        icon.png
        background.png
    drawable-hdpi/
        icon.png
        background.png
```

W podanym przykładzie zaprezentowane zostały dwa katalogi do przechowywania grafik:
- res/drawable – zasoby graficzne używane zawsze, niezależnie od rozdzielczości ekranu,
- res/drawable-hdpi – grafika dla urządzeń z ekranami o wysokiej rozdzielczości.

Oprócz dwóch wymienionych istnieją jeszcze inne, dedykowane do różnego rodzaju urządzeń:
- /res/drawable-ldpi – obrazy dla urządzeń z ekranami o niskiej rozdzielczości,
- /res/drawable-mdpi – obrazy dla urządzeń z ekranami o średniej rozdzielczości,
- /res/drawable-xhdpi – grafika dla urządzeń z ekranami o ekstra wysokiej rozdzielczości.

6. W trakcie uruchamiania aplikacji system automatycznie określa, które zasoby najlepiej spełniają wymagania danego urządzenia. Platforma Android zawsze wczytuje najbardziej precyzyjnie

określony zasób, najlepiej dostosowany do danej sytuacji. Jeśli zasób alternatywny nie istnieje, używany jest zasób domyślny. Do dobrych praktyk programistycznych zaliczamy dbanie o to, aby dla wszystkich zasobów alternatywnych istniały odpowiadające im zasoby domyślne, dzięki czemu niezależnie od konfiguracji urządzenia jakaś wersja zasobu zawsze zostanie wczytana. *Więcej informacji:* https://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources.

7. Warto jednak pamiętać, że próby dostosowywania zasobów do wszelkich możliwości mogą znacznie powiększyć wielkość projektu aplikacji. Ma to też wpływ na wydajność działania aplikacji. Wskazane jest zatem projektowanie jak najbardziej elastycznych i skalowalnych zasobów domyślnych.
8. Należy przestrzegać ustalonej konwencji nazewnictwa → tabela konfiguracyjna.
9. Określając wymiary i położenie elementów interfejsu użytkownika nie należy używać precyzyjnych, ustalonych wartości. Zamiast tego lepiej rozmieszczać je względem innych elementów.
10. Wielkości należy określać w jednostkach elastycznych, takich jak dp (wymiary układów i grafik, piksele wyrażane względem ekranu 160 dpi, niezależne od gęstości ekranu) lub sp (wielkość tekstu, piksele niezależne od skali).
11. Poprawa wizualnej atrakcyjności aplikacji może się odbywać poprzez zastosowanie zdefiniowanych motywów i stylów. Motyw stosowany jest w aplikacji (lub aktywności) jako całości – stosowany jest do określania spójnego wyglądu aplikacji. Style służą do określania wyglądu konkretnych widoków lub ich grup. *Więcej informacji*: https://developer.android.com/guide/topics/ui/look-and-feel/themes.html.
12. Kolory stosowane w aplikacji definiowane są w pliku res/values/colors.xml. Przechowujemy wartości RGB zaczynające się od znaku #. Dostępne są różne formaty zapisu wartości RGB:
    a. #RGB (np. #F00) – 12-bitowa wartość koloru,
    b. #ARGB (np. #8F00) – 12-bitowa wartość koloru i dodatkowo zdefiniowana przezroczystość,
    c. #RRGGBB (np. #FF00FF) – 24-bitowa wartość koloru,
    d. #AARRGGBB (np. #80FF00FF) - 24-bitowa wartość koloru i określona przezroczystość.
13. Możliwe jest określanie różnych kolorów lub zasobów graficznych, które będą stosowane zależnie od stanu kontrolki. Do tego celu stosowany jest specjalny typ zasobów: <selector>. Przykład: ustalenie listy kolorów dla różnych stanów przycisku: gdy jest nieaktywny, włączony lub wciśnięty. *Więcej informacji*: https://developer.android.com/guide/topics/resources/color-list-resource.html.
14. Łańcuchy znaków definiujemy w pliku xml w katalogu /res/values. Domyślnie plik nosi nazwę strings.xml. Każdy łańcuch znakowy definiowany jest przy pomocy znacznika <string> jako para nazwa-wartość. Możliwe jest tworzenie łańcuchów znaków, których formę można zmieniać w zależności od tego, czy będą występować w liczbie pojedynczej, czy mnogiej. Stosowany jest w tym celu znacznik <plurals>:

```
<plurals name="numberOfSelectedItems">
    <item quantity="one">Zaznaczono jeden produkt</item>
    <item quantity="other">Zaznaczono %d produktów</item>
</plurals>
```

Łańcuch formatujący %d w powyższym pozwala na wyświetlenie konkretnej liczby zaznaczonych produktów. Do pracy z takimi zasobami w kodzie aplikacji służy metoda *getQuantityString()*. Wymaga ona przekazania trzech parametrów:

- zasobu łańcuchowego,

- wartości określającej liczbę, która informuje, która wersja słowa powinna być wyświetlona,
- konkretnej wartości liczbowej, która zostanie wyświetlona w miejscu wystąpienia sekwencji %d.

Więcej informacji: https://developer.android.com/guide/topics/resources/string-resource.html#Plurals.

15. Stosując odpowiednie nazwy plików i uzupełniając je tłumaczeniami łańcuchów znakowych, można zapewnić umiędzynarodowienie aplikacji. Standardowo do nazwy pliku strings.xml dodawany jest dwuliterowy kod (identyfikator) państwa. Odpowiednia wersja tłumaczeń dobierana jest **automatycznie** na podstawie konfiguracji urządzenia. *Więcej informacji:* https://developer.android.com/guide/topics/resources/localization.html.
16. Dostęp do zasobów można uzyskać na dwa sposoby:
    a. Z poziomu xml:

    ```
    @[<package_name>:]<resource_type>/<resource_name>
    ```

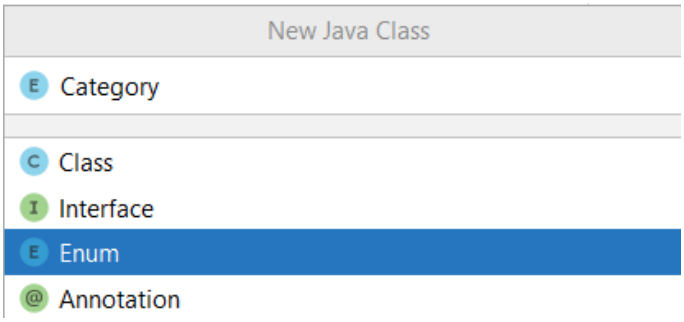
    b. Z poziomu kodu Java:

    ```
    [<package_name>.]R.<resource_type>.<resource_name>
    ```

17. Wiele aplikacji posiada menu. Menu to zasoby podobne do układów. Definiowane są w plikach xml. Wyróżniamy następujące rodzaje menu:
    a. **Options menu** – standardowy zbiór akcji dostępnych w pasku aplikacji (*app bar)*. Elementy menu powinny być powiązane z danym widokiem (np. wyszukiwanie kontaktu z listy wyświetlanej w danej aktywności).
    b. **Context menu** – ruchome menu pojawiające się, gdy użytkownik przyciśnie przez dłuższy czas wybrany element widoku. Jest ściśle powiązane z wybranym elementem.
    c. **Popup menu** – menu przypisane do konkretnego fragmentu widoku, posiadające elementy, które umożliwiają wykonywanie czynności powiązanych z wybranym obszarem.
18. Do stworzenia *options menu* konieczne są przynajmniej następujące kroki:
    a. Utworzenie odpowiednich elementów typu layout:
        - Element nadrzędny <menu>
        - Zagnieżdżone elementy <item>
    b. Zaimplementowanie (nadpisanie) dwóch podstawowych metod w klasie aktywności:
        - onCreateOptionsMenu – przypisanie menu do danej aktywności, konfiguracja, dodanie nowych elementów, zmiana ustawień istniejących;
        - onOptionsItemSelected – obsługa zdarzenia wybrania elementu menu przez użytkownika; odbywa się standardowo przy pomocy instrukcji switch.
19. Omawiając temat projektowania interfejsów użytkownika dostosowanych do różnych wymagań (poprzez definiowanie zasobów alternatywnych) warto też wspomnieć o rekomendowanej praktyce stosowania fragmentów. Służą one uniezależnieniu projektu ekranów od kodu konkretnych klas aktywności i pozwalają zapewnić elastyczność sposobu obsługi aplikacji. Klasy aktywności mogą dowolnie łączyć i stosować komponenty interfejsu użytkownika bądź zachowania, tworząc z ich pomocą znacznie bardziej elastyczny interfejs użytkownika. Stosując fragmenty można być przygotowanym do obsługi nowych urządzeń, które w przyszłości pojawią się na rynku. *Więcej informacji*: https://developer.android.com/guide/components/fragments.html.

TREŚĆ ZADANIA

1. **Cel:** zapoznanie się z metodami zarządzania zasobami. W ramach tego zadania aplikacja do zapisywania zadań do zrobienia (*zadanie nr 3: TodoApp*) zostanie udoskonalona pod względem wizualnym oraz wzbogacona o nowe funkcjonalności. Zostaną wprowadzone m.in. następujące zmiany:
    a. Dodanie przy każdym zadaniu z listy elementu checkbox, który będzie pozwalał na oznaczenie zadania jako wykonane.
    b. Wyświetlanie zadania, które zostało wykonane jako przekreślone.
    c. Dodanie do każdego zadania na liście ikony, która będzie wskazywała na kategorię zadania.
    d. Dodanie nowych kontrolek: listy rozwijanej oraz pola wyboru daty.
    e. Dodanie menu.
    f. Implementacja funkcjonalności dodawania nowych zadań do listy.
    g. Dynamiczne wyświetlanie liczby zaległych zadań w pasku narzędzi.
    h. Umiędzynarodowienie.
    i. Zdefiniowanie stylów.
2. Dodaj typ wyliczeniowy (*Enum*) o nazwie Category. Uzupełnij go o dwie kategorie zadań (np. studia, dom).

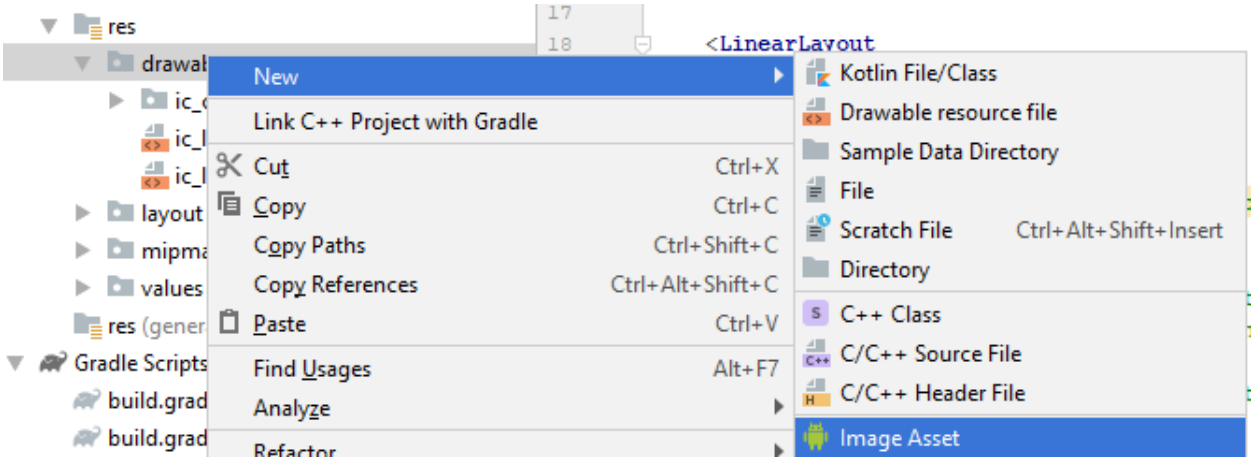


3. Dodaj do klasy Task pole typu Category o takiej samej nazwie oraz metody dostępowe.
4. W konstruktorze w klasie TaskStorage dodaj ustawianie kategorii:

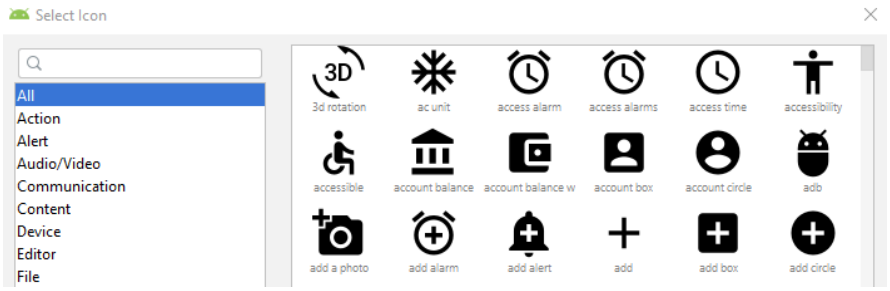
```
if (i % 3 == 0) {
    task.setCategory(Category.STUDIES);
} else {
    task.setCategory(Category.HOME);
}
```

5. W celu wzbogacenia widoku listy o ikony wyświetlane przy każdym elemencie, otwórz plik *list_item_task.xml*. Dodaj do niego kontrolkę *ImageView* w taki sposób, aby wyświetlała się po prawej stronie dotychczasowych elementów (pól tekstowych). Ikona będzie wskazywała kategorię zadania.

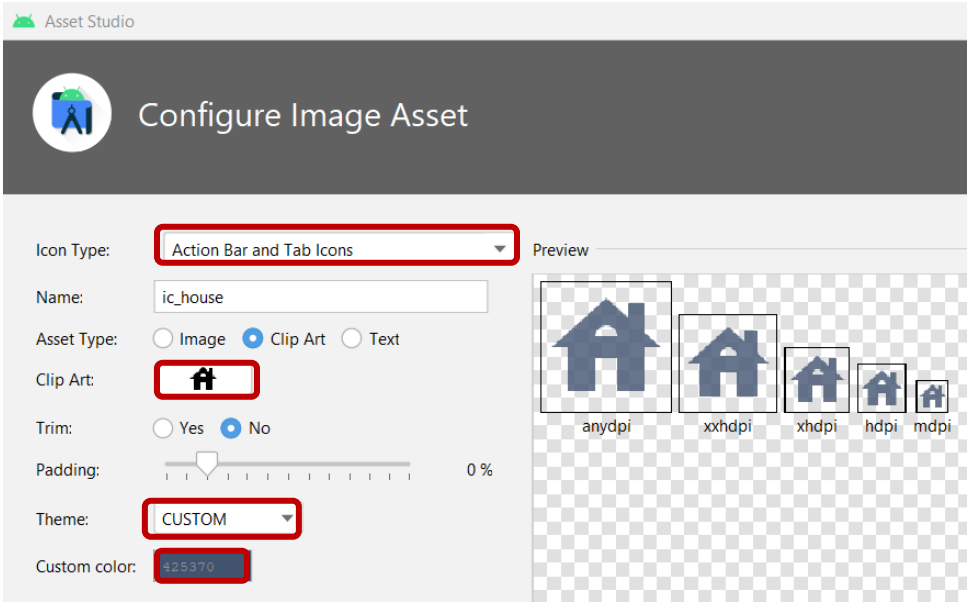
6. Wybierz ikony, które będą się wyświetlały w przygotowanej kontrolce *ImageView* w zależności od wybranej kategorii zadania. Użyj w tym celu wbudowanego narzędzia *Android Asset Studio*. Narzędzie to pozwala na tworzenie i dostosowywanie do własnych potrzeb elementów graficznych stanowiących ikony.



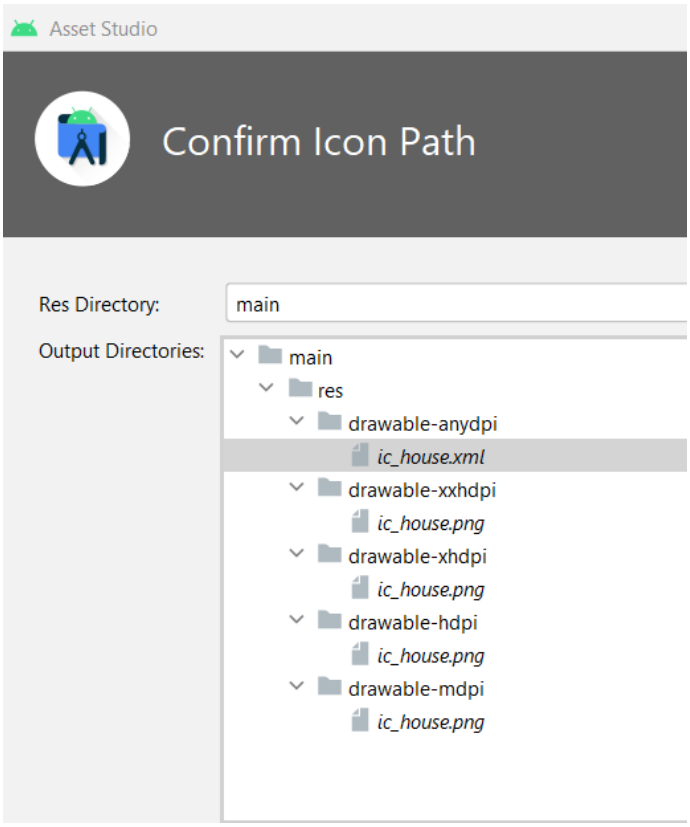
Wybierz z listy dwie ikony, które będą pasowały do wyświetlenia kategorii zadań: studia i dom (każda z ikon musi zostać dodana oddzielnie).



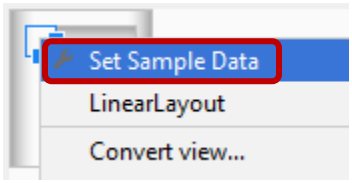
Ustaw odpowiednio parametry ikony, zgodnie z własnymi preferencjami.

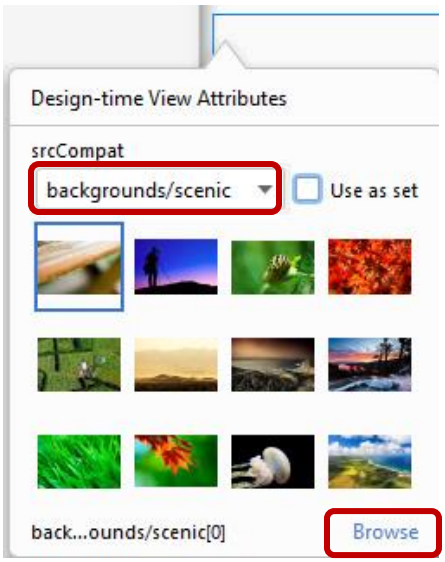


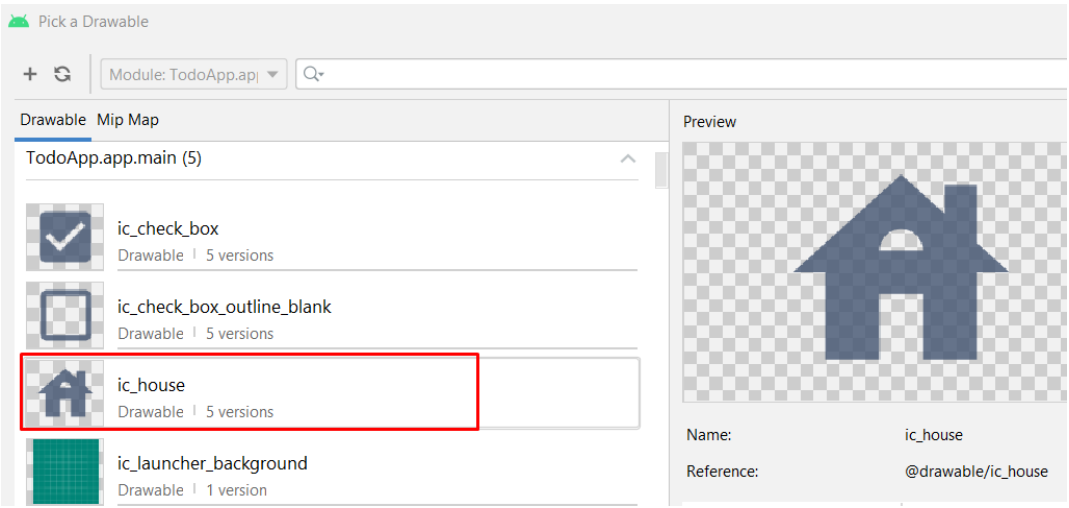
Zauważ, że narzędzie generuje automatycznie zestaw ikon dla każdego typu gęstości ekranu.



7. Ustaw domyślną ikonę dla zadania klikając prawym przyciskiem na pole *ImageView* widoczne w zakładce *Design* widoku *list_item_task.xml* i wybierz opcję *Set Sample Data*:







8. Dodaj nowy komponent (*ImageView*) w klasie *TaskHolder* (analogicznie do innych elementów widoku). Zapewnij, że ikona będzie dostosowana do wartości pola *task.category*.

```
if (task.getCategory().equals(Category.HOME)) {
    iconImageView.setImageResource(R.drawable.ic_house);
} else {
    iconImageView.setImageResource(R.drawable.ic_university);
}
```

9. Do pliku *list_item_task.xml* dodaj również element typu *CheckBox*. Kontrolka powinna znajdować się po lewej stronie od nazwy zadania.
10. Zaimplementuj ustawianie wartości dodanego pola *CheckBox* w klasie *TaskHolder* (pobranie kontrolki z widoku i ustawienie jej wartości zgodnie z wartością odpowiadającego jej pola z klasy *Task*).
11. W klasie *TaskAdapter* w metodzie *onBindViewHolder()* dodaj następujący kod:

```
CheckBox checkBox = holder.getCheckBox();
checkBox.setChecked(tasks.get(position).isDone());
checkBox.setOnCheckedChangeListener((buttonView, isChecked) ->
        tasks.get(holder.getBindingAdapterPosition()).setDone(isChecked));
```

12. Zmodyfikuj istniejący kod w taki sposób, aby po oznaczeniu zadania jako wykonane jego nazwa na liście stawała się przekreślona. Dodatkowo zapewnij, że wyświetlana na liście nazwa każdego zadania będzie się mieściła w jednej linii. Zadbaj o poprawne obsłużenie sytuacji, gdy nazwa będzie dłuższa – w miejscu uciętego tekstu powinny pojawiać się trzy kropki.
13. Popraw widok edycji zadania - w pliku *fragment_task.xml* zamień kontrolkę typu *Button* o id = task_date na następującą:

```
<EditText
    android:id="@+id/task_date"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:clickable="false"
    android:cursorVisible="false"
    android:focusable="false"
    android:focusableInTouchMode="false"
    android:autofillHints=""
    android:inputType="date" />
```

14. W klasie *TaskFragment* dodaj obsługę *DatePickera* – okna wyboru daty, które będzie się pojawiało przy kliknięciu w przygotowany element typu *EditText*:

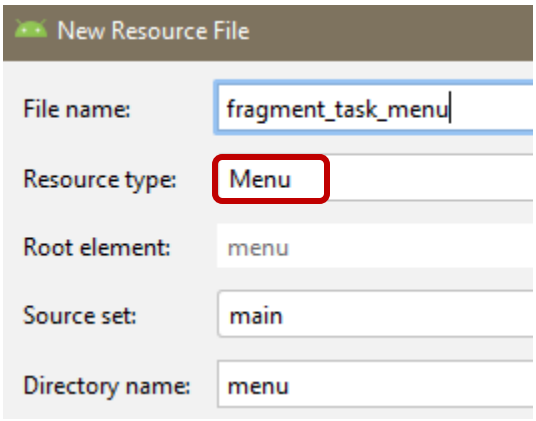
```
private final Calendar calendar = Calendar.getInstance();

dateField = view.findViewById(R.id.task_date);
DatePickerDialog.OnDateSetListener date = (view12, year, month, day) -> {
    calendar.set(Calendar.YEAR, year);
    calendar.set(Calendar.MONTH, month);
    calendar.set(Calendar.DAY_OF_MONTH, day);
    setupDateFieldValue(calendar.getTime());
    task.setDate(calendar.getTime());
};
dateField.setOnClickListener(view1 ->
        new DatePickerDialog(getContext(), date, calendar.get(Calendar.YEAR),
                calendar.get(Calendar.MONTH), calendar.get(Calendar.DAY_OF_MONTH))
                .show());
setupDateFieldValue(task.getDate());

private void setupDateFieldValue(Date date) {
    Locale locale = new Locale( language: "pl",  country: "PL");
    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "dd.MM.yyyy", locale);
    dateField.setText(dateFormat.format(date));
}
```

15. Zaktualizuj plik z łańcuchami znakowymi (*strings.xml*):

```
<string name="task_add_new">Nowe zadanie</string>
<string name="show_subtitle">Pokaż podtytuł</string>
<string name="hide_subtitle">Ukryj podtytuł</string>
<string name="subtitle_format">%1$d zadań</string>
```

16. Następnym krokiem będzie dodanie menu w pasku narzędzi. Zazwyczaj prawa górna część paska narzędzi jest zarezerwowana na przyciski akcji (polecenia menu). W ramach tego zadania menu będzie składało się z jednego przycisku, który pozwoli na dodawanie nowych zadań.

17. W celu utworzenia definicji menu w xml, kliknij prawym przyciskiem na katalog *res* i wybierz polecenie *New > Android resource file*. Następnie wybierz jako typ zasobu *menu*:



18. W wygenerowanym pliku *fragment_task_menu.xml* dodaj nowy element menu, który będzie służył do dodawania zadań na listę. Dobierz odpowiednią ikonkę.

```
<item
    android:id="@+id/new_task"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/task_add_new"
    app:showAsAction="ifRoom|withText"/>
```

19. Zwróć uwagę na atrybut *showAsAction* – odnosi się on do tego, czy dany element menu będzie widoczny na pasku narzędzi, czy ukryty w tzw. *menu przepełnienia* (ang. *overflow menu*). Dodanie dwóch wartości tego atrybutu w powyższym przykładzie oznacza, że uwzględnione zostały dwie sytuacje:
    a. Jeśli na pasku narzędzi wystarczy miejsca, to zostanie wyświetlona ikona z opisem;
    b. Jeśli na pasku narzędzi nie będzie wystarczająco dużo miejsca, wyświetlona zostanie sama ikona;
    c. W przypadku całkowitego braku miejsca na pasku narzędzi, dany element zostanie przeniesiony do ukrytego menu przepełnienia.

    Ikona przepełnienia to trzy pionowe kropki widoczne po prawej stronie paska narzędzi. Dobrą praktyką jest umieszczanie na pasku narzędzi jedynie często używanych przycisków akcji (elementów menu). Najlepiej więc unikać opcji *always* atrybutu *showAsAction*.
20. Uruchom aplikację w trybie standardowym oraz zmieniając orientację urządzenia – porównaj sposób wyświetlania dodanej ikony menu.
21. Standardowo menu jest zarządzane z poziomu aktywności, w której system wywołuje metodę *onCreateOptionsMenu()*. W przypadku aplikacji *TodoList* wszystkie tego typu czynności oddelegowane są do fragmentu – to tam zostanie więc dodana obsługa menu.
22. W klasie *TaskListFragment* nadpisz metodę zwrotną *onCreateOptionsMenu()*:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_task_menu, menu);
}
```

23. Za wywołanie powyższej metody odpowiada *FragmentManager*. To on przechwytuje od aktywności wywołanie zwrotne metody *onCreateOptionsMenu* z systemu operacyjnego. Nie dzieje się tak jednak domyślnie – trzeba jawnie poinformować menedżera *FragmentManager*, że *TaskListFragment* powinien otrzymać wywołanie metody *onCreateOptionsMenu()*. Można to zrobić poprzez nadpisanie metody *onCreate()* we wspomnianym fragmencie i wywołanie w niej następującej metody:

```
@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}
```

24. W klasie *TaskStorage* dodaj metodę *addTask()*:

```
public void addTask(Task task) {
    tasks.add(task);
}
```

25. W klasie *TaskListFragment* nadpisz kolejną metodę obsługującą menu – *onOptionsItemSelected()*:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_task:
            Task task = new Task();
            TaskStorage.getInstance().addTask(task);
            Intent intent = new Intent(getActivity(), MainActivity.class);
            intent.putExtra(TaskListFragment.KEY_EXTRA_TASK_ID, task.getId());
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Powyższa metoda uruchamia istniejącą aktywność *MainActivity* zawierającą fragment *TaskFragment*, który wyświetla szczegóły zadania – w tym wypadku będą one puste, gdyż zadanie jest nowe. Po uzupełnieniu wymaganych pól i cofnięciu do poprzedniego widoku, nowe zadanie powinno pojawić się na liście. Uwaga: w konstruktorze klasy *Task* dodaj ustawianie domyślnej kategorii.

26. W widoku służącym do edycji zadania (*fragment_task.xml*) dodaj pole wyboru kategorii (użyj kontrolki typu *Spinner*). Dodaj również jej obsługę w klasie *TaskFragment* w metodzie *onCreateView()*:

```
categorySpinner = view.findViewById(R.id.task_category);
categorySpinner.setAdapter(new ArrayAdapter<>(this.getContext(), android.R.layout.simple_spinner_item, Category.values()));
categorySpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        task.setCategory(Category.values()[position]);
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
    }
});
categorySpinner.setSelection(task.getCategory().ordinal());
```

27. Dodaj definicję kolejnego elementu menu (plik *fragment_task_menu.xml*), który będzie służył do wyświetlania podtytułu.

```
<item
    android:id="@+id/show_subtitle"
    android:title="@string/show_subtitle"
    app:showAsAction="ifRoom"/>
```

28. Nowy element będzie służył do wyświetlania informacji o liczbie aktualnie znajdujących się na liście niewykonanych zadań. Konieczne więc jest dodanie w *TaskListFragment* metody, która będzie odpowiedzialna za zliczanie zaległych zadań:

```
public void updateSubtitle() {
    TaskStorage taskStorage = TaskStorage.getInstance();
    List<Task> tasks = taskStorage.getTasks();
    int todoTasksCount = 0;
    for (Task task : tasks) {
        if (!task.isDone()) {
            todoTasksCount++;
        }
    }
    String subtitle = getString(R.string.subtitle_format, todoTasksCount);
    AppCompatActivity appCompatActivity = (AppCompatActivity) getActivity();
    appCompatActivity.getSupportActionBar().setSubtitle(subtitle);
}
```

29. Dodaj wywołanie utworzonej metody (*updateSubtitle()*) w metodzie *onOptionsItemSelected()*:

```
case R.id.show_subtitle:
    updateSubtitle();
    return true;
```

30. Sprawdź działanie aplikacji po wprowadzonej zmianie.
31. Zaimplementuj również opcję ukrywania podtytułu. W tym celu dodaj w klasie *TaskListFragment* zmienną zapamiętującą, czy podtytuł jest widoczny:

```
private boolean subtitleVisible;
```

32. Dodaj następujący kod do metody *onCreateOptionsMenu()*:

```
MenuItem subtitleItem = menu.findItem(R.id.show_subtitle);
if (subtitleVisible) {
    subtitleItem.setTitle(R.string.hide_subtitle);
} else {
    subtitleItem.setTitle(R.string.show_subtitle);
}
```

33. Zaktualizuj również metodę *onOptionsItemSelected()*:

```
case R.id.show_subtitle:
    subtitleVisible = !subtitleVisible;
    getActivity().invalidateOptionsMenu();
    updateSubtitle();
    return true;
```

Metoda *invalidateOptionsMenu()* służy do wymuszenia rekonstrukcji przycisków akcji menu.

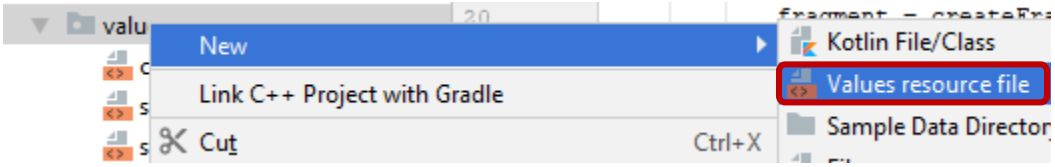
34. Dodaj również w metodzie *updateSubtitle()* ukrywanie wyświetlanego napisu podtytułu w zależności od wartości zmiennej *subtitleVisible*:

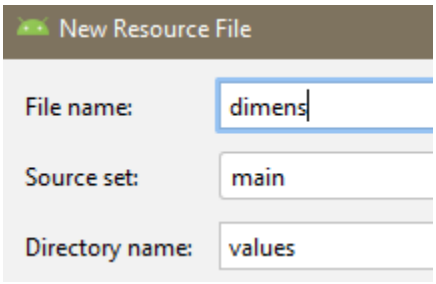
```
String subtitle = getString(R.string.subtitle_format, todoTasksCount);
if (!subtitleVisible) {
    subtitle = null;
}
AppCompatActivity appCompatActivity = (AppCompatActivity) getActivity();
appCompatActivity.getSupportActionBar().setSubtitle(subtitle);
```

35. Aby poprawnie zaktualizować wyświetlaną w podtytule liczbę zadań po dodaniu nowego zadania, dodaj na końcu metody *updateView()* w klasie *TaskListFragment* metodę *updateSubtitle()*.

36. Zadbaj o to, aby podtytuł nie znikał po zmianie orientacji urządzenia.

37. Utwórz plik *dimens.xml* w katalogu res > values:





38. W pliku tym zdefiniuj kilka wielkości, które zostaną użyte jako marginesy, odstępy lub wielkości czcionek, np.:

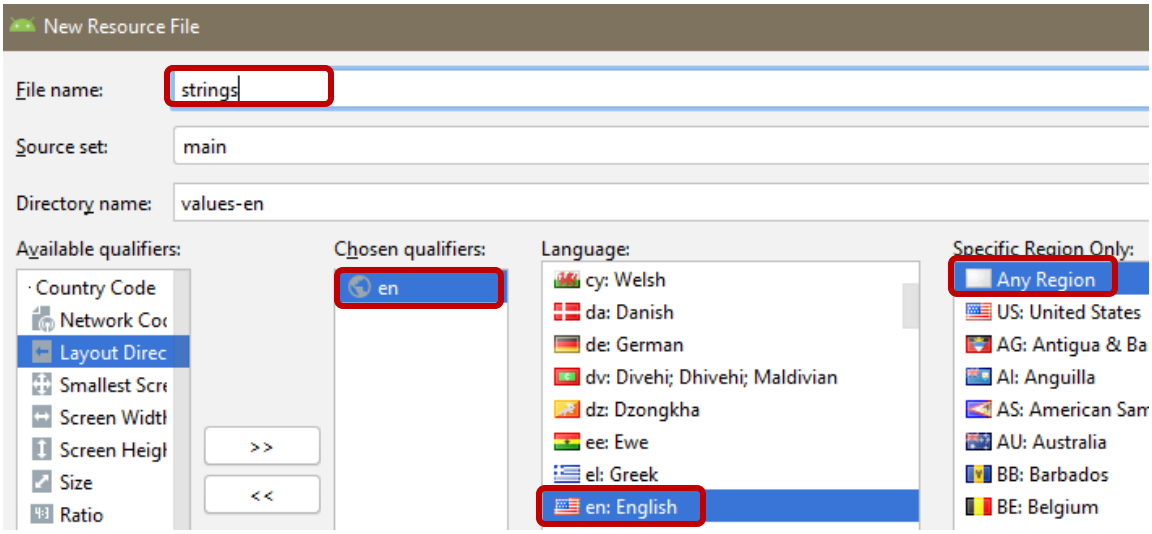
```
<resources>
    <dimen name="task_item_padding">10dp</dimen>
```

39. Wykorzystaj zdefiniowane wielkości do ustawienia marginesów, odstępów oraz wielkości czcionek tak, aby udoskonalić warstwę wizualną aplikacji. Przykład:

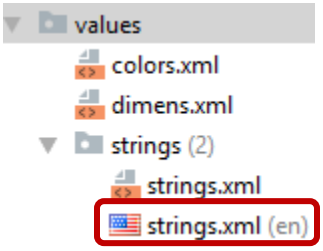
```
<ImageView
    android:id="@+id/task_undone_icon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="@dimen/task_item_padding"
```

40. Zadbaj o to, aby wszystkie wyświetlane w aplikacji łańcuchy znakowe były zapisane w pliku *strings.xml*.

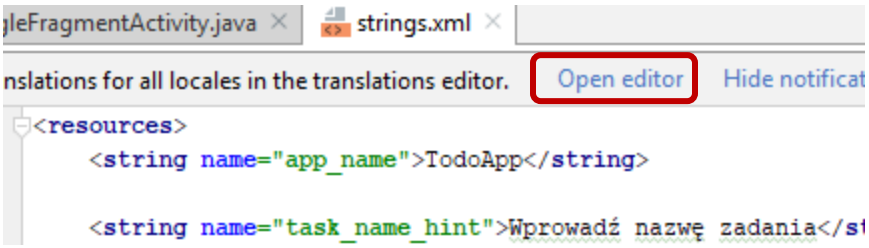
41. Dodaj tłumaczenia wszystkich łańcuchów znakowych. W tym celu utwórz nowy plik zasobów klikając prawym przyciskiem myszy na katalog *res* > *values*, a następnie wybierając opcję *New* > *Values resource file*:



42. Zauważ, że w katalogu *values* pojawił się nowy katalog, zawierający wewnątrz dwa pliki *strings.xml*: domyślny zawierający łańcuchy znakowe w języku polskim oraz dodane tłumaczenia na język angielski.



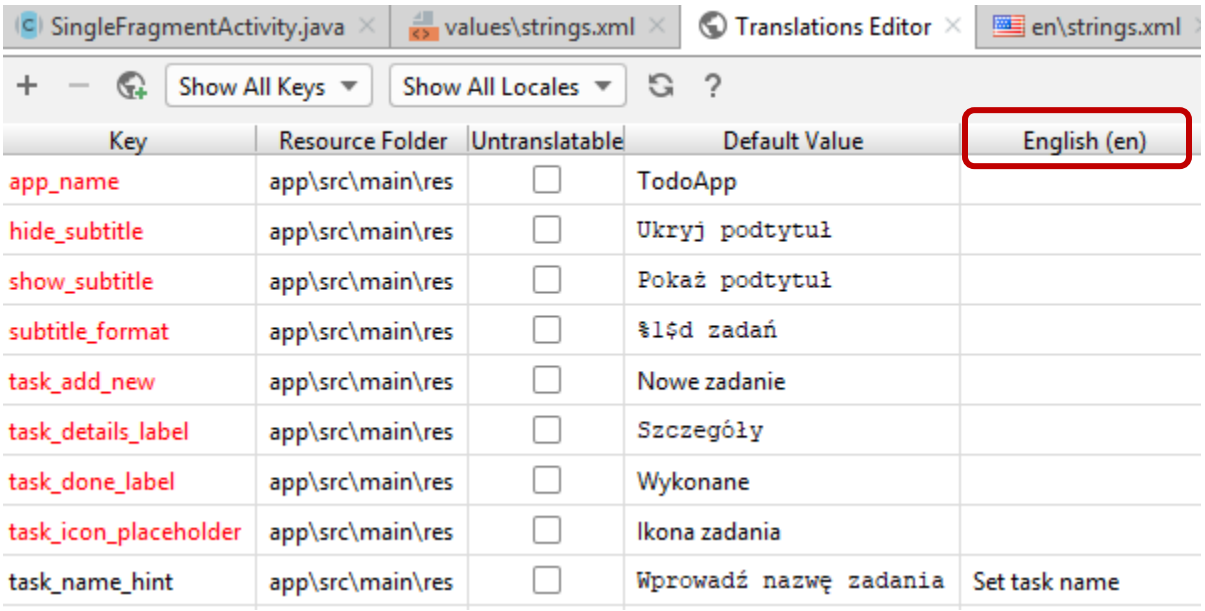
43. **Uzupełnij dodany plik** – możesz to zrobić korzystając z dedykowanego edytora:



Dodaj przynajmniej jedno tłumaczenie w pliku z wersją angielską:

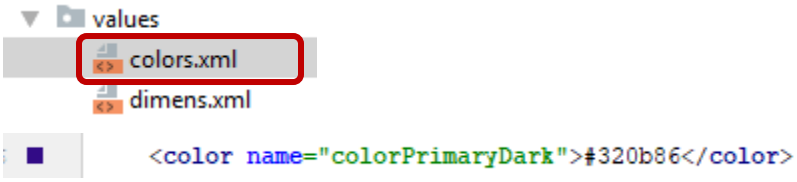
```
<resources>
    <string name="task_name_hint">Set task name</string>
```

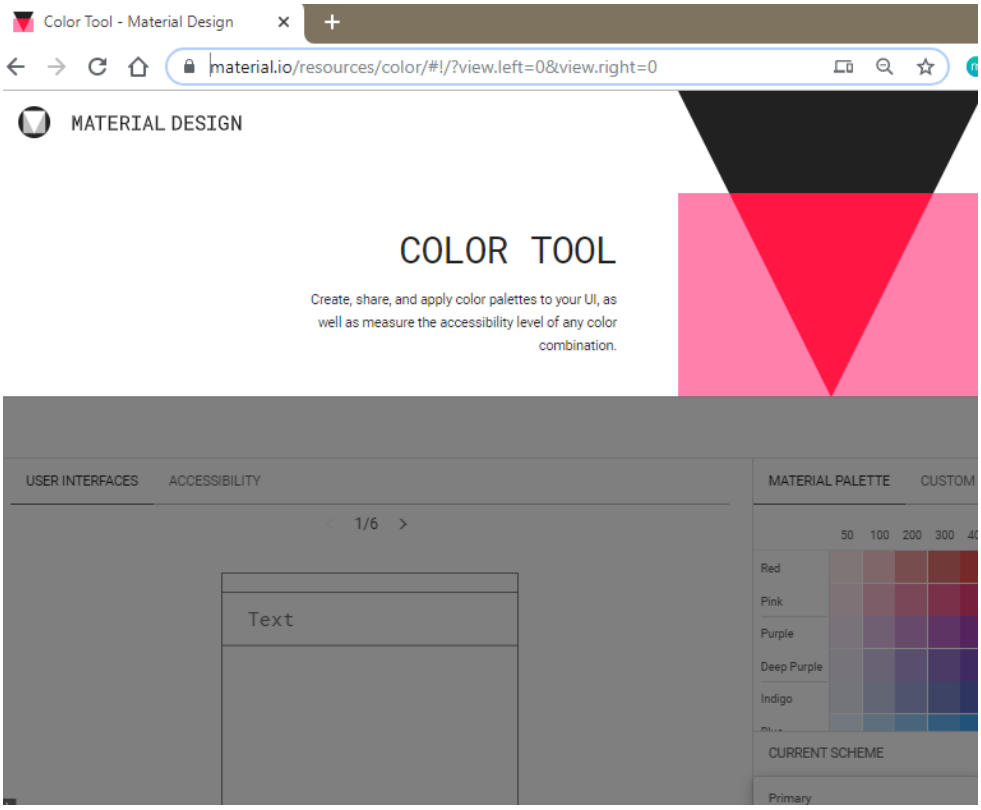
Po tym edytor powinien zostać zaktualizowany o dodatkową kolumnę dla języka angielskiego.



Key	Resource Folder	Untranslatable	Default Value	English (en)
app_name	app\src\main\res	☐	TodoApp	
hide_subtitle	app\src\main\res	☐	Ukryj podtytuł	
show_subtitle	app\src\main\res	☐	Pokaż podtytuł	
subtitle_format	app\src\main\res	☐	%1$d zadań	
task_add_new	app\src\main\res	☐	Nowe zadanie	
task_details_label	app\src\main\res	☐	Szczegóły	
task_done_label	app\src\main\res	☐	Wykonane	
task_icon_placeholder	app\src\main\res	☐	Ikona zadania	
task_name_hint	app\src\main\res	☐	Wprowadź nazwę zadania	Set task name

44. Zdefiniuj kilka kolorów w pliku *colors.xml* (do wyboru kolorystyki aplikacji najlepiej skorzystaj z narzędzia https://material.io/resources/color/#!/?view.left=0&view.right=0)

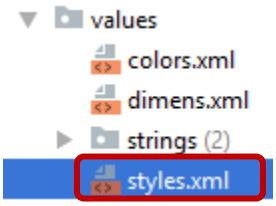




45. Przypisz kolory wybranym elementom widoku, np.:

```
<TextView
    style="?android:listSeparatorTextViewStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Nazwa"
    android:textColor="@color/colorPrimaryDark"/>
```

46. W pliku *styles.*xml utwórz dwa style. Każdy z nich powinien konfigurować ustawienia **co najmniej 5** właściwości kontrolek.



47. Przykładowe przypisanie utworzonego stylu:

```
<TextView
    android:id="@+id/task_item_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="@style/TextFieldsStyle"
```

48. Przetestuj działanie aplikacji. W celu sprawdzenia poprawności działania umiędzynarodowienia, zmień ustawienia regionalne w urządzeniu lub emulatorze.