TEMAT: Tworzenie interfejsu użytkownika: fragmenty, listy, adaptery.

WPROWADZENIE

1. Fragmenty to obiekty kontrolera, które mogą być wyznaczane przez powiązaną aktywność do wykonania określonego zadania. Każdy fragment, podobnie jak aktywność, posiada cykl życia oraz definicję widoku. Widok fragmentu może być wstawiany w widoku aktywności w specjalnie do tego przeznaczonym obszarze – kontenerze na fragment (obszarów takich może być kilka). Zazwyczaj fragmenty wykorzystywane są do tworzenia bardziej przyjaznego użytkownikowi interfejsu użytkownika oraz do wielokrotnego wykorzystania podobnego kodu.
2. Fragment musi posiadać aktywność zarządzającą (hostującą). Aby spełniać taką rolę, aktywność musi:
    a. Posiadać w definicji swojego układu miejsce (kontener) na hostowany fragment,
    b. Zarządzać cyklem życia instancji fragmentu.
3. Fragmenty posiadają metody cyklu życia odpowiadające metodom aktywności. Dzięki temu mogą współpracować z określonymi stanami cyklu życia aktywności zarządzającej.
4. **Zapamiętaj:** metody cyklu życia fragmentu są wywoływane przez aktywność hostującą dany fragment (a nie przez system operacyjny).
5. Listy to nieodłączny element interfejsu użytkownika bardzo wielu aplikacji. Rekomendowaną metodą tworzenia list jest aktualnie korzystanie z kontenera *RecyclerView*. Kontener ten zarządza poszczególnymi elementami listy oraz dba o to, aby elementy składowe widoku listy (czyli wiersze będące obiektami typu *View*) były tworzone tylko dla tych obiektów, które są widoczne na ekranie. W ten sposób unikamy nadmiernego zużycia pamięci. Przy tworzeniu listy typu *RecyclerView* tworzymy też powiązane obiekty: *Adapter* i *ViewHolder*.
6. *ViewHolder* przechowuje elementy widoku, które są wyświetlane w jednym wierszu listy (np. pole tekstowe i obrazek).
7. *Adapter* jest odpowiedzialny za tworzenie obiektów *ViewHolder* oraz powiązanie tych obiektów z danymi, które znajdują się w modelu i tworzą daną listę (są jej elementami).

TREŚĆ ZADANIA

1. **Cel:** zapoznanie się z ważnymi i wielokrotnie wykorzystywanymi w różnych aplikacjach elementami tworzącymi widok – listami oraz fragmentami. Rezultatem wykonania zadania będzie wstępna wersja aplikacji do zapisywania zadań do zrobienia (TodoApp).
2. Stwórz nową aplikację z pustą aktywnością (*Empty Activity*) o domyślnej nazwie *MainActivity*.
3. Dodaj klasę reprezentującą model, czyli zadanie do zrobienia: *Task*. Przy ustawianiu identyfikatorów zadań skorzystaj z predefiniowanej klasy *UUID* (*Universally Unique Identifier).*

```
public class Task {
    private UUID id;
    private String name;
    private Date date;
    private boolean done;

    public Task() {
        id = UUID.randomUUID();
        date = new Date();
    }
}
```

4. W kodzie xml widoku przypisanego do utworzonej aktywności zdefiniuj kontener na fragment:

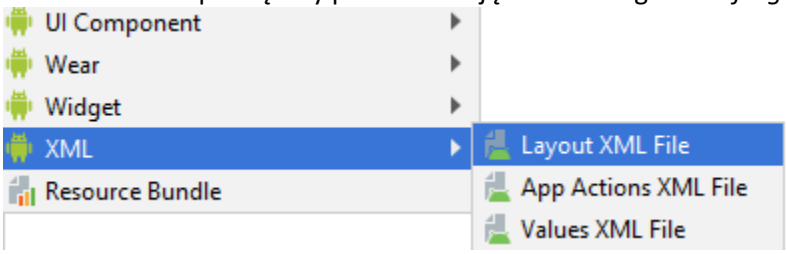
```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

   Zmień domyślny typ układu na *FrameLayout*. Jak widać, kontener nie posiada odwołania do żadnej aktywności – dzięki temu może być uniwersalny.

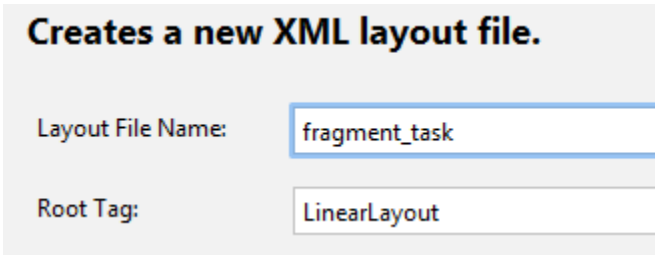
5. Stwórz nowy fragment *TaskFragment* poprzez dodanie nowej klasy Java:

```
public class TaskFragment extends Fragment {
```

6. Utwórz również powiązany plik z definicją widoku fragmentu *fragment_task.xml*:



   Wybierz *LinearLayout* jako element główny układu:



7. W konfiguracji widoku fragmentu (korzystając z edytora lub kodu xml) ustaw domyślny margines w całym układzie na *20dp* i orientację pionową. Dodaj następujące kontrolki:
   a. *TextView*
   b. *EditText*
   c. *TextView*
   d. *Button*
   e. *CheckBox*

Kod powinien przypominać następujący:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="20dp"
    android:orientation="vertical">

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/name_label" />

    <EditText
        android:id="@+id/task_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/name_hint"/>

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/details_label"/>

    <Button
        android:id="@+id/task_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <CheckBox
        android:id="@+id/task_done"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/done_label"/>

</LinearLayout>
```

8. Dodaj łańcuchy znakowe:

```
<string name="name_hint">Wpisz nazwę zadania</string>
<string name="name_label">Nazwa</string>
<string name="details_label">Szczegóły</string>
<string name="done_label">Wykonane</string>
```

9. W klasie fragmentu *TaskFragment* nadpisz następujące metody:
    a. *onCreate()* – dodaj tworzenie nowego obiektu typu *Task*.
    b. *onCreateView()* – zamiast domyślnego kodu dodaj pobieranie i zwracanie definicji widoku zawartej w pliku *fragment_task.xml*:

```
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup con
    return inflater.inflate(R.layout.fragment_task, container, attachToRoot: false);
}
```

Dodaj też w tym miejscu obsługę kontrolek (pól tekstowych i przycisków): pobranie, implementację metod nasłuchujących, ustawienie wartości (poniżej znajdują się tylko fragmenty wymaganego kodu – do samodzielnego uzupełnienia):

```
nameField = view.findViewById(R.id.task_name);
nameField.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {

    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        task.setName(s.toString());
    }

    @Override
    public void afterTextChanged(Editable s) {

    }
});

dateButton.setText(task.getDate().toString());
dateButton.setEnabled(false);

doneCheckBox.setChecked(task.isDone());
doneCheckBox.setOnCheckedChangeListener((buttonView, isChecked) -> task.setDone(isChecked));
```

10. W klasie aktywności *MainActivity* w metodzie *onCreate()* dodaj do istniejącego kodu pobieranie obiektu *FragmentManager*, który jest odpowiedzialny za zarządzanie fragmentami i dodawanie ich widoków do hierarchii widoków aktywności. Dodaj transakcję tworzącą fragment.

```
FragmentManager fragmentManager = getSupportFragmentManager();
Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);

if (fragment == null) {
    fragment = new TaskFragment();
    fragmentManager.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
}
```

Transakcje są wykorzystywane do dodawania, usuwania, przyłączania, odłączania oraz podmiany fragmentów znajdujących się na liście. *FragmentManager* przechowuje stos transakcji fragmentów i zarządza nimi. Pozwala to w niezwykle elastyczny sposób tworzyć i modyfikować wygląd ekranu podczas działania aplikacji.

11. Stwórz klasę typu Singleton do scentralizowanego przechowywania listy zadań. Klasa powinna mieć nazwę *TaskStorage*. Dodaj w niej pole zawierające listę zadań oraz dwie metody dostępowe do tej listy:
    - Zwracającą całą listę zadań,
    - Zwracającą pojedyncze zadanie o podanym jako parametr identyfikatorze.

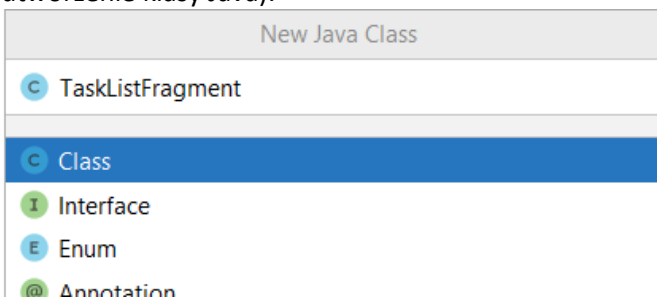
    Dodaj również w prywatnym konstruktorze kod generujący automatycznie przykładowe zadania i dodaje je na listę (liczba zadań > 100).

    Uwaga: tworzenie klasy typu *Singleton* należy wykonać zgodnie z poznanymi regułami implementacji wzorców projektowych.

    Fragment przykładowego rozwiązania:

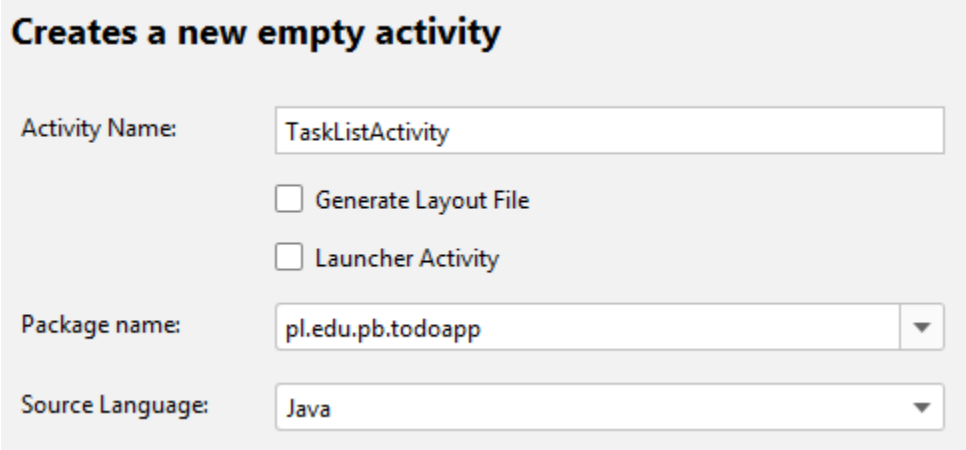
```
public class TaskStorage {
    private static final TaskStorage taskStorage = new TaskStorage();

    private final List<Task> tasks;

    public static TaskStorage getInstance() { return taskStorage; }

    private TaskStorage() {
        tasks = new ArrayList<>();
        for (int i = 1; i <= 150; i++) {
            Task task = new Task();
            task.setName("Pilne zadanie numer " + i);
            task.setDone(i % 3 == 0);
            tasks.add(task);
        }
    }
```

12. Utwórz nowy fragment o nazwie *TaskListFragment* (podobnie jak poprzednio: poprzez utworzenie klasy Java).




```
public class TaskListFragment extends Fragment {
```

13. Stwórz nową aktywność, która będzie przechowywała fragment *TaskListFragment*. Nie wybieraj opcji automatycznego generowania powiązanego pliku układu (layout):



Tym razem skorzystamy z istniejącego już układu – tego, który był przygotowany w trakcie tworzenia pierwszej aktywności. Jak pamiętasz, nie był on przypisany do żadnej konkretnej aktywności, dzięki czemu może być wielokrotnie wykorzystany w różnych aktywnościach. Nie jest tu też zadeklarowany żaden konkretny fragment, jedynie kontener do przechowywania dowolnego fragmentu.
Przypomnij sobie, jak wyglądało przypisanie kontekstu do kodu xml układu (widoku) w poprzednim zadaniu, dotyczącym Quizu:

```
tools:context=".MainActivity"
```

14. Dodatkowo okazuje się, że kod zarządzający fragmentami z pierwszej aktywności będzie **powtarzalny** – również druga aktywność powinna w podobny sposób implementować metodę *onCreate()*. W celu zredukowania powtarzalnego kodu utwórz klasę abstrakcyjną aktywności, która będzie zawierała powtarzalny kod i nie będzie miała przypisanego widoku.

```
public abstract class SingleFragmentActivity extends AppCompatActivity {
```

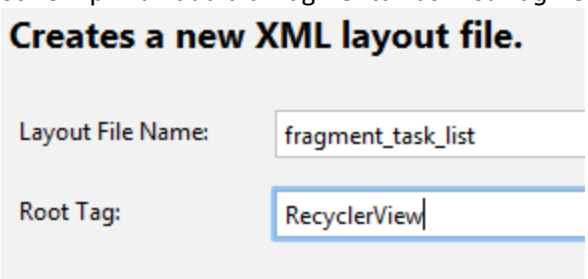
15. Do stworzonej klasy abstrakcyjnej **przenieś** kod z metody *onCreate()* z pierwszej aktywności (z klasy *MainActivity*), zamieniając linię tworzącą konkretny fragment na wywołanie metody abstrakcyjnej *createFragment()*:

```
protected abstract Fragment createFragment();
```

16. W **obydwu klasach aktywności** dodaj do definicji dziedziczenie po klasie abstrakcyjnej *SingleFragmentActivity* i zaimplementuj metodę *createFragment()*.
17. Zmień konfigurację aplikacji w ten sposób, aby jako pierwsza uruchamiała się aktywność z listą zadań (*TaskListActivity*).
18. Dodaj zależność (*dependency*), która umożliwi korzystanie z *RecyclerView*:

```
implementation 'androidx.recyclerview:recyclerview:1.2.1'
```

19. Stwórz plik układu dla fragmentu *TaskListFragment*:



Dodaj identyfikator.
Plik układu powinien przypominać poniższy:

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.recyclerview.widget.RecyclerView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/task_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</androidx.recyclerview.widget.RecyclerView>
```

20. W kodzie fragmentu *TaskListFragment* nadpisz metodę *onCreateView()* i dodaj w niej powiązanie z utworzonym plikiem układu *fragment_task_list.xml* (zamiast wywołania metody *super.onCreateView* klasy nadrzędnej). Pobierz również odwołanie do układu *RecyclerView* tworząc zmienną tego typu i używając metody *findViewById()*:

```
View view = inflater.inflate(R.layout.fragment_task_list, container, attachToRoot: false);
recyclerView = view.findViewById(R.id.task_recycler_view);
```

21. Dodaj także kod ustawiający rodzaj ułożenia elementów na liście – w tym wypadku będzie to ułożenie liniowe (pionowa lista - *LinearLayoutManager*):

```
recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
```

22. Utwórz widok dla pojedynczego elementu listy (*list_item_task.xml*). Układ powinien być typu *LinearLayout*. Dodaj do niego dwa pola tekstowe *TextView* - do wyświetlania nazwy zadania oraz daty. Nadaj im identyfikatory: *task_item_name, task_item_date*.
23. Stwórz klasę wewnętrzną *ViewHolder* w klasie *TaskListFragment*:

```
private class TaskHolder extends RecyclerView.ViewHolder {

    public TaskHolder(LayoutInflater inflater, ViewGroup parent) {
        super(inflater.inflate(R.layout.list_item_task, parent, attachToRoot: false));
    }
}
```

24. Stwórz analogicznie klasę wewnętrzną *TaskAdapter*. Zaimplementuj wymagane metody i dodaj konstruktor:

```
private class TaskAdapter extends RecyclerView.Adapter<TaskHolder> {
    private List<Task> tasks;

    public TaskAdapter(List<Task> tasks) {
        this.tasks = tasks;
    }

    @NonNull
    @Override
    public TaskHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
        return new TaskHolder(layoutInflater, parent);
    }

    @Override
    public void onBindViewHolder(@NonNull TaskHolder holder, int position) {

    }

    @Override
    public int getItemCount() {
        return tasks.size();
    }
}
```

25. W klasie *TaskListFragment* połącz utworzony adapter z kontenerem *RecyclerView*. Zaimplementuj w tym celu metodę *updateView()* i wywołaj ją z poziomu metody *onCreateView()* klasy *TaskListFragment* tuż przed wywołaniem *return*:

```
private void updateView() {
    TaskStorage taskStorage = TaskStorage.getInstance();
    List<Task> tasks = taskStorage.getTasks();

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

26. W klasie *TaskHolder* dodaj opcję nasłuchiwania wciśnięcia dowolnego elementu listy poprzez implementację interfejsu *OnClickListener*:

```
private class TaskHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
```

27. W klasie *TaskHolder* dodaj deklaracje zmiennych odpowiadających elementom widoku pojedynczego zadania z listy, w konstruktorze ustaw obiekt nasłuchujący oraz zaimplementuj metodę *bind(),* która będzie dowiązywała wartości pól konkretnego zadania do elementów widoku *item_list_task.xml*:

```
public TaskHolder(LayoutInflater inflater, ViewGroup parent) {
    super(inflater.inflate(R.layout.list_item_task, parent,  attachToRoot: false));
    itemView.setOnClickListener(this);

    nameTextView = itemView.findViewById(R.id.task_item_name);
    dateTextView = itemView.findViewById(R.id.task_item_date);
}

public void bind(Task task) {
    this.task = task;
    nameTextView.setText(task.getName());
    dateTextView.setText(task.getDate().toString());
}
```

28. Wywołaj powyższą metodę w nadpisanej metodzie *onBindViewHolder()* w klasie *TaskAdapter:*

```
@Override
public void onBindViewHolder(@NonNull TaskHolder holder, int position) {
    Task task = tasks.get(position);
    holder.bind(task);
}
```

29. Stwórz wywołanie aktywności ze szczegółami zadania (*MainActivity*) po kliknięciu wybranego elementu listy – w tym celu zaimplementuj metodę *onClick()* w klasie *TaskHolder*:

```
@Override
public void onClick(View v) {
    Intent intent = new Intent(getActivity(), MainActivity.class);
    intent.putExtra(KEY_EXTRA_TASK_ID, task.getId());
    startActivity(intent);
}
```

30. W metodzie *onCreate()* klasy *TaskFragment* pobierz przekazaną wartość id wybranego zadania (w które kliknął użytkownik) i **zamiast** tworzyć nowe, puste zadanie, pobierz korzystając z *TaskStorage* istniejące zadanie o podanym id.

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

    UUID taskId = (UUID) getArguments().getSerializable(ARG_TASK_ID);
    task = TaskStorage.getInstance().getTask(taskId);
}
```

31. Dodaj do kodu fragmentu *TaskFragment* metodę statyczną tworzącą nową jego instancję:

```
public static TaskFragment newInstance(UUID taskId) {
    Bundle bundle = new Bundle();
    bundle.putSerializable(ARG_TASK_ID, taskId);
    TaskFragment taskFragment = new TaskFragment();
    taskFragment.setArguments(bundle);
    return taskFragment;
}
```

32. W metodzie *onCreateView()* klasy *TaskFragment* ustaw wartości wyświetlanych pól na te, które posiada wybrane zadanie – wykorzystując metody *setText* oraz *setChecked* w zależności od typu pola.
33. W klasie *MainActivity* zmodyfikuj metodę *createFragment()* w następujący sposób:

```
@Override
protected Fragment createFragment() {
    UUID taskId = (UUID) getIntent().getSerializableExtra(TaskListFragment.KEY_EXTRA_TASK_ID);
    return TaskFragment.newInstance(taskId);
}
```

34. W klasie zawierającej listę (*TaskListFragment*) nadpisz metodę *onResume()*:

```
@Override
public void onResume() {
    super.onResume();
    updateView();
}
```

35. Wykorzystując wiedzę z poprzednich zajęć, dodaj opcję aktualizowania nazwy wybranego zadania tak, aby po cofnięciu do aktywności zawierającej listę zadań wyświetlała się ustawiona w widoku szczegółów nazwa.