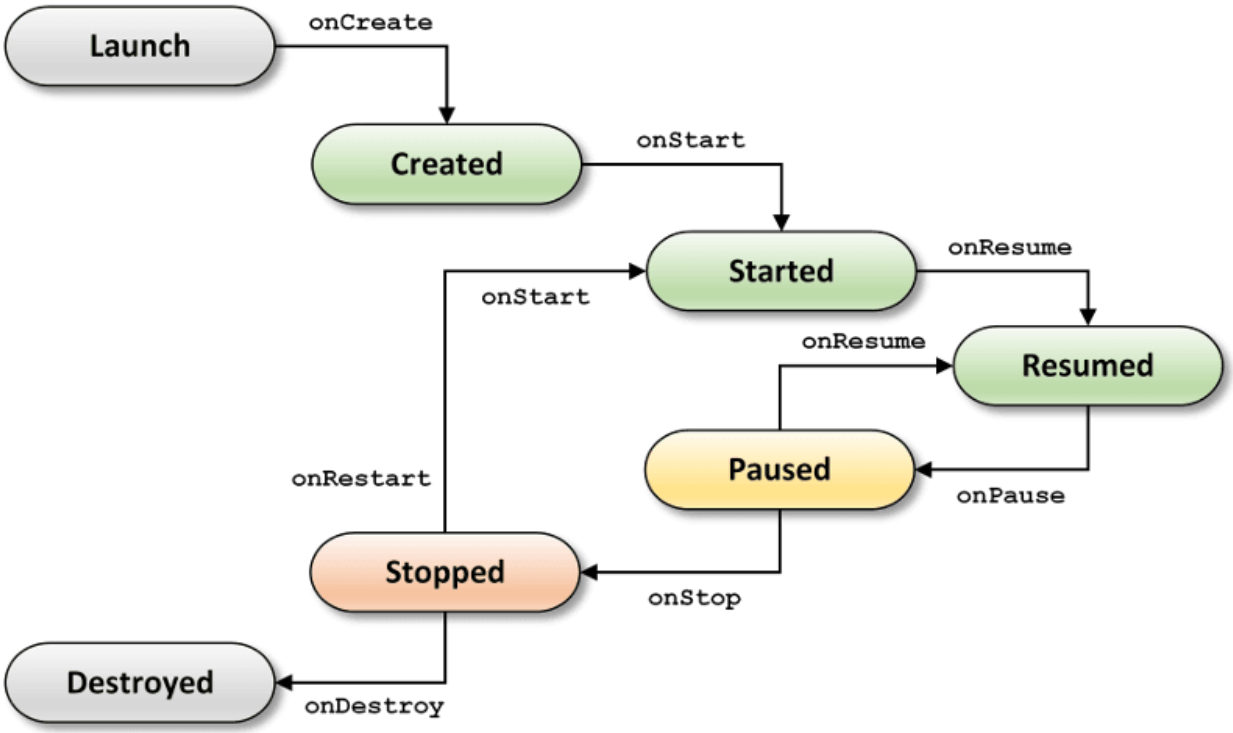
TEMAT: Cykl życia aplikacji. Przełączanie między aktywnościami.

WPROWADZENIE

CYKL ŻYCIA AKTYWNOŚCI

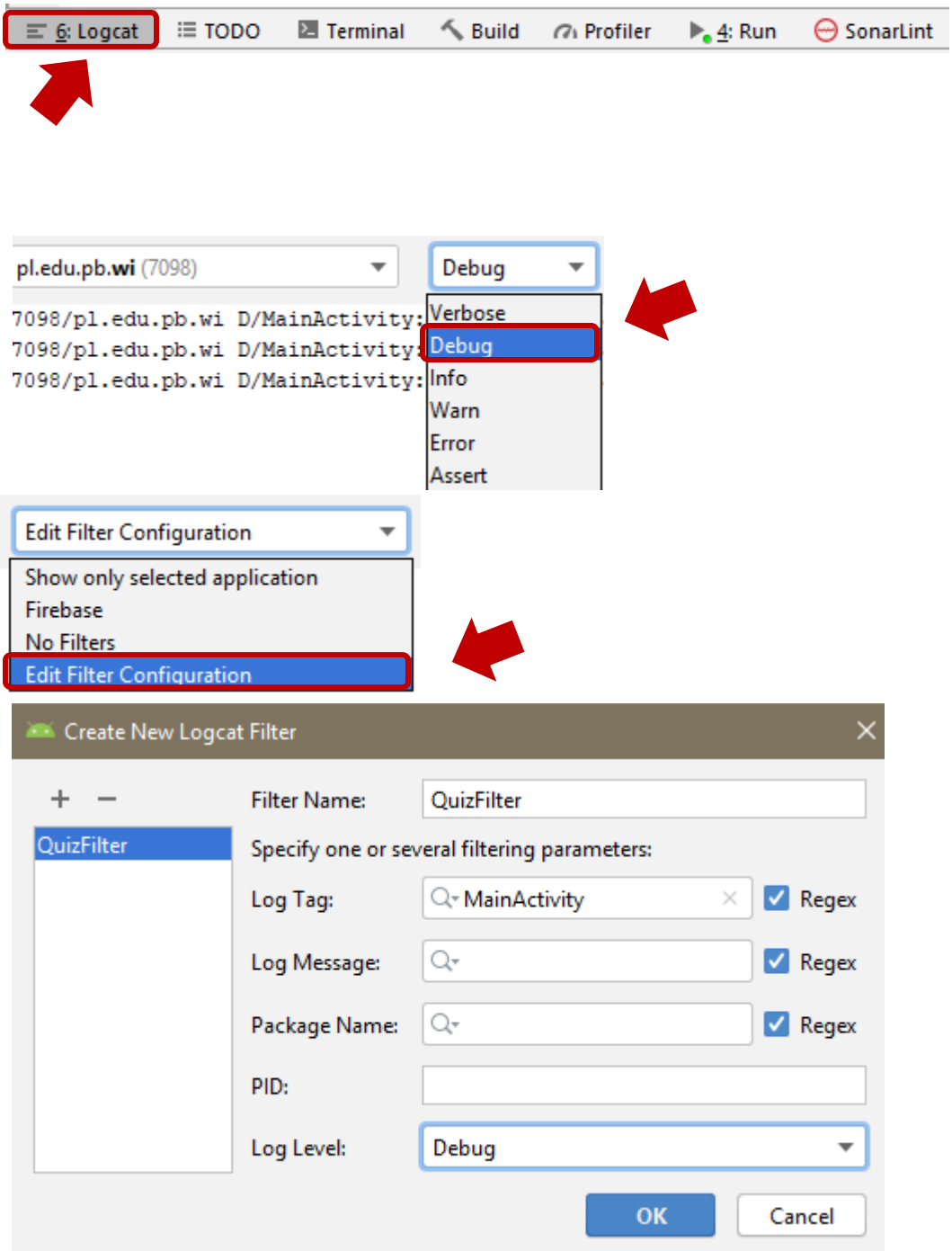
1. Każda instancja obiektu klasy *Activity* posiada swój cykl życia.
2. W cyklu życia aktywności możemy wyróżnić cztery główne stany:
    a. Uruchomiona/wznowiona – użytkownik aktualnie korzysta z aktywności, aktywność jest widoczna na pierwszym planie, jest przechowywana w pamięci;
    b. Zawieszona – aktywność jest częściowo (lub nawet całkowicie) widoczna, jednak nie jest na pierwszym planie (np. w przypadku, gdy korzystamy z opcji dzielenia ekranu na wiele jednocześnie otwartych okien aplikacji i zajmujemy się w danym momencie inną aktywnością, a zawieszona aktywność jest również widoczna), jest przechowywana w pamięci;
    c. Zatrzymana – aktywność pozostaje w pamięci, nie jest widoczna i nie działa na pierwszym planie (można znów zacząć z niej korzystać np. przywracając ją z listy ostatnio używanych aplikacji);
    d. Nieistniejąca – aktywność jest zniszczona, czyli niewidoczna (więc również nie jest na pierwszym planie) oraz całkowicie usunięta z pamięci.
3. Momenty, w których aktywność zmienia swój stan mogą zostać odpowiednio obsłużone poprzez implementację metod zwrotnych cyklu życia aktywności (*activity lifecycle callbacks*).



4. Jedna z tych metod (uruchamiana jako pierwsza po włączeniu aktywności, czyli *onCreate()*) musi być zawsze nadpisana, pozostałe zaś nie są obowiązkowe do napisania.
5. System Android automatycznie wywołuje metody cyklu życia w odpowiednich momentach, związanych ze zmianami stanu aktywności.
6. Zapamiętaj: nigdy nie wywołuj samodzielnie metod cyklu życia aktywności!
7. Procesy zawierające aktywności działające na pierwszym planie lub zawieszone mają wyższy priorytet niż inne procesy. Stąd w przypadku konieczności zwolnienia zasobów, system rozpoczyna usuwanie z pamięci procesów (a więc też aktywności), które mają niższy priorytet, czyli są w innych stanach.

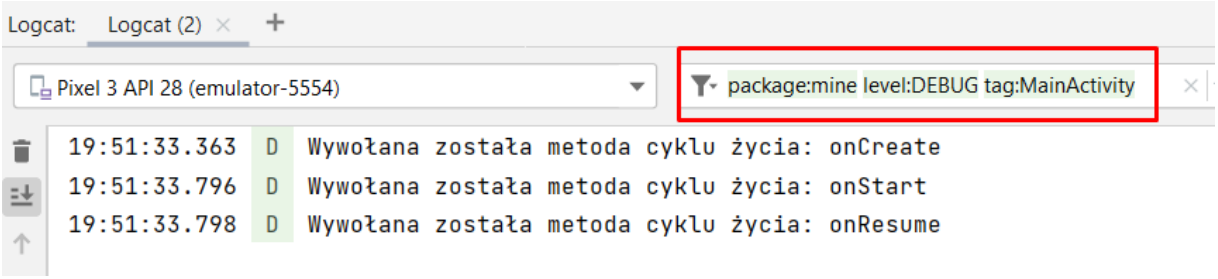
TREŚĆ ZADANIA

1. **Cel:** zapoznanie się z cyklem życia aktywności, zapisanie stanu aplikacji przy obracaniu ekranu, zapoznanie się z mechanizmem nawigacji między aktywnościami.
2. Aby wykonać zadanie skorzystaj z aplikacji stworzonej na poprzednich zajęciach. Zadanie będzie polegało na modyfikacji zaimplementowanego kodu, aby poszerzyć funkcjonalności aplikacji i poprawnie obsłużyć sytuacje, które mogą powodować błędne działanie aplikacji.
3. Pierwsza część zadania będzie obejmowała analizę wywołań metod cyklu życia w zależności od interakcji użytkownika z aplikacją.
4. W tym celu nadpisz metody cyklu życia głównej aktywności, która została stworzona na poprzednich zajęciach (skrót *Ctrl + o*). Metoda *onCreate* została już napisana – jak zapewne pamiętasz, jest obowiązkowa do implementacji. Pozostałe metody do nadpisania:
    a. *onStart*
    b. *onResume*
    c. *onPause*
    d. *onStop*
    e. *onDestroy*
5. W każdej z metod cyklu życia dodaj zapisywanie do logów, że została wywołana. Skorzystaj z metody *Log.d()*.
   Pamiętaj, że wywołania metod z klasy nadrzędnej (*super*) zawsze muszą występować w pierwszej linii nadpisywanej metody.
6. Uruchom aplikację i zbadaj, jakie metody cyklu życia są wywoływane przy wykonywaniu następujących czynności:
    - Po pierwszym uruchomieniu aplikacji
    - Po wciśnięciu przycisku *Cofnij*
    - Po ponownym uruchomieniu aplikacji
    - Po wciśnięciu przycisku ekranu głównego *Home*
    - Po powrocie do aplikacji z listy ostatnio używanych aplikacji
7. Logi pojawiają się w panelu Logcat. Zazwyczaj pojawia się ich tam bardzo dużo, więc warto skorzystać z opcji tworzenia filtrów (starsze wersje Android Studio):



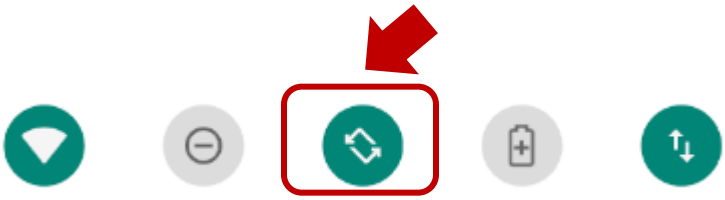
Zaznaczamy opcję Log Level = *Debug*, gdyż wpisując kod *Log.d()* wybraliśmy DEBUG jako poziom zapisywanego do logów komunikatu. Inne poziomy wywoływane są przez inne metody, np. *Log.e()* - błędy (ERROR), *Log.w()* – ostrzeżenia (WARNING), *Log.i()* – informacje (INFO).

W najnowszej wersji Android Studio dodano odświeżoną wersję dziennika komunikatów Logcat. Zmieniło się m.in. formatowanie wyświetlanych logów oraz wyszukiwanie. Poniżej znajduje się przykład filtrowania oczekiwanych logów w nowej wersji:



8. Wyniki powyższej analizy zapisz do pliku tekstowego, **który powinien być przesłany w ramach rozwiązania zadania**. Zaznacz wyraźnie (poprzez opis), po której czynności pojawiły się dane wywołania metod cyklu życia w logach.
9. Sprawdź, jaki wpływ na cykl życia aktywności ma zmiana orientacji ekranu. W tym celu uruchom aplikację, przejdź do drugiego pytania klikając przycisk *Następne* i obróć urządzenie.
   W emulatorze klikamy ikonę: 

   Aplikacja musi się faktycznie obrócić wraz z urządzeniem, zadbaj, aby obracanie ekranu było włączone:



10. Co się stało po obróceniu ekranu? Które pytanie się wyświetla? **Dodaj do pliku z zapisanymi logami również te, które zostały wypisane po zmianie orientacji urządzenia**.
11. Zadbaj, aby aplikacja zachowywała swój stan przy zniszczeniu głównej aktywności (wywołaniu metody *onDestroy*). Dzięki temu zapamiętywane będzie ostatnie wyświetlane pytanie i po ponownym uruchomieniu właśnie to pytanie wyświetli się jako pierwsze. W tym celu nadpisz metodę *onSaveInstanceState*:

```
public class MainActivity extends AppCompatActivity {
    private static final String KEY_CURRENT_INDEX = "currentIndex";

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    Log.d(QUIZ_TAG,  msg: "Wywołana została metoda: onSaveInstanceState");
    outState.putInt(KEY_CURRENT_INDEX, currentIndex);
}
```

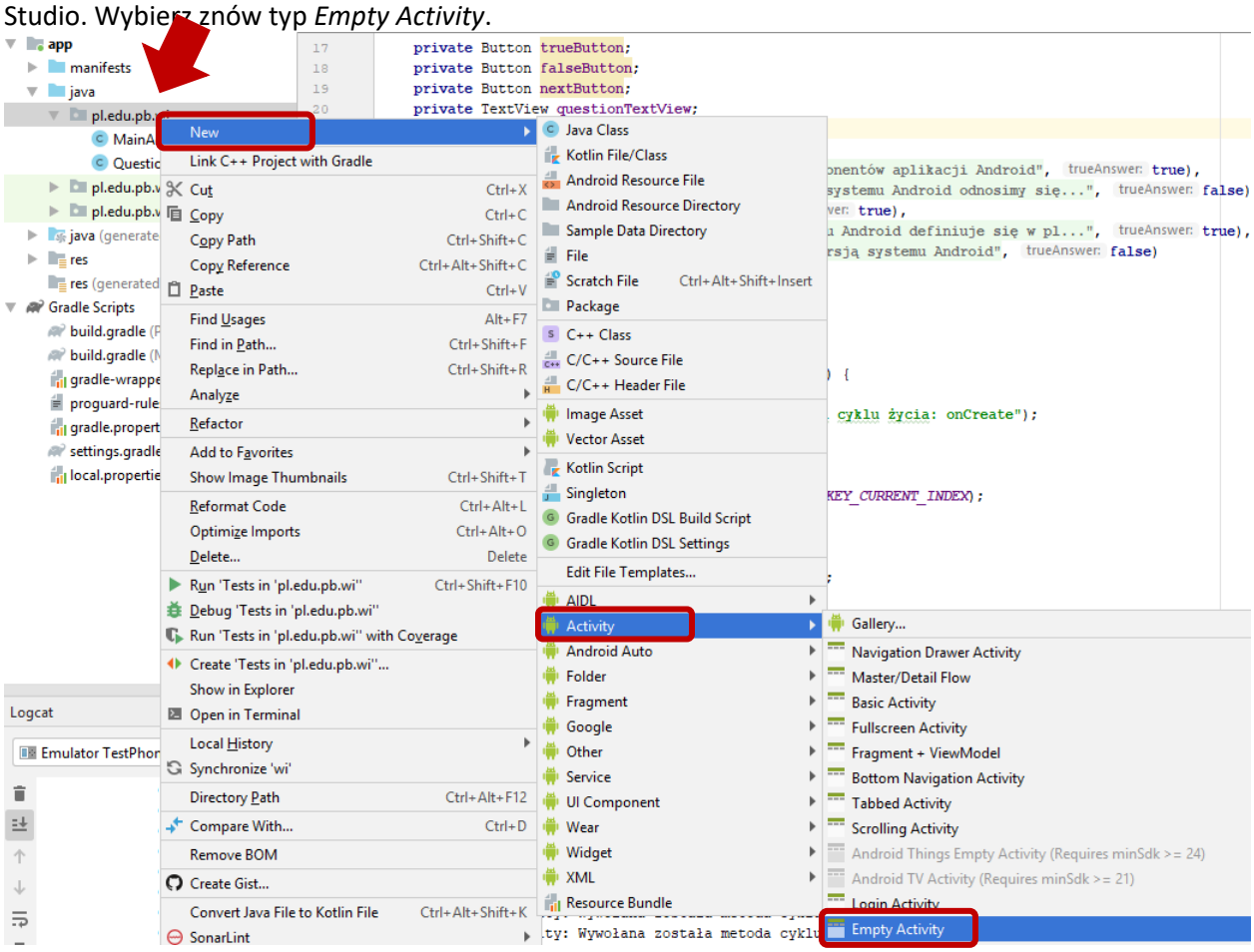
Metoda ta jest wywoływana przed wywołaniem metody *onStop()*, z wyjątkiem przypadku, w którym wciśnięty został przycisk *Cofnij*. W obiekcie *Bundle* można zapisywać wartości określonych typów i potem odnosić się do nich za pomocą zdefiniowanych kluczy.

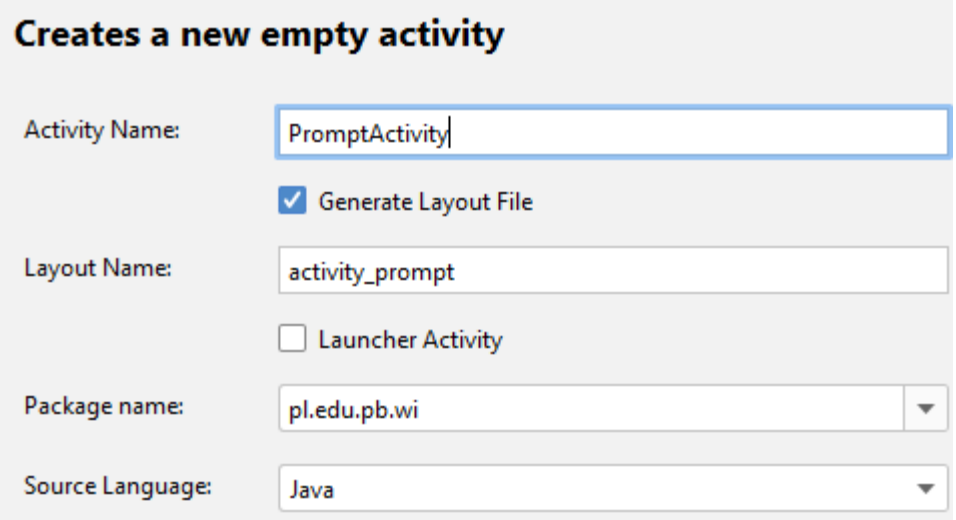
12. Pobranie zapisanych wartości (stanu aplikacji) odbywa się w metodzie *onCreate*. Tam też zaimplementuj odczytanie wartości indeksu ostatnio wyświetlanego pytania ze struktury *Bundle*:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(QUIZ_TAG,  msg: "Wywołana została metoda cyklu życia: onCreate");
    setContentView(R.layout.activity_main);

    if (savedInstanceState != null) {
        currentIndex = savedInstanceState.getInt(KEY_CURRENT_INDEX);
    }
```

13. Zrestartuj aplikację i powtórz kroki z punktu 9 – które pytanie wyświetli się tym razem? **Dodaj odpowiedź do pliku tekstowego z logami cyklu życia aplikacji.**
14. Stwórz nową aktywność, która będzie wyświetlała podpowiedzi do pytań quizu (*w podanym tu przykładzie dla uproszczenia wyświetlana jest po prostu odpowiedź na pytanie*). Skorzystaj z kreatora automatycznego tworzenia aktywności, dostępnego domyślnie w środowisku Android Studio. Wybierz znów typ *Empty Activity*.





15. Sprawdź zawartość pliku *AndroidManifest.xml*. Czy pojawił się tam wpis dotyczący nowej aktywności?
16. W edytorze widoku nowej aktywności (plik *res/layout/activity_prompt.xml*) dodaj pole tekstowe z napisem "Czy na pewno chcesz zobaczyć odpowiedź?" oraz przycisk z napisem "Pokaż odpowiedź".
17. Dodaj również pole tekstowe, w którym po wciśnięciu przycisku "Pokaż odpowiedź", pojawi się odpowiedź. Pole na razie nie może mieć przypisanego tekstu, gdyż będzie on zależny od aktualnego pytania. Stąd zastosuj specjalny typ atrybutu: *tools:text*. Pozwala on na nadpisanie zawartości atrybutu *text* w trakcie działania programu. Wpisany domyślny tekst nie będzie więc wyświetlany (pozwala jedynie sprawdzić, jak pole tekstowe będzie się prezentowało w widoku aktywności).

```
<TextView
    android:id="@+id/answer_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="30dp"
    tools:text="Odpowiedź"/>
```

18. Do widoku pierwszej aktywności dodaj przycisk "Podpowiedź" i zaimplementuj w klasie aktywności powiązany z przyciskiem *listener*.
19. W metodzie nasłuchującej *onClick()* dodaj polecenie (intencję) wywołania drugiej aktywności i przekaż poprawną odpowiedź stosując metodę *putExtra()*:

```
public class MainActivity extends AppCompatActivity {
    public static final String KEY_EXTRA_ANSWER = "pl.edu.pb.wi.quiz.correctAnswer";

promptButton.setOnClickListener((v) → {
        Intent intent = new Intent( packageContext: MainActivity.this, PromptActivity.class);
        boolean correctAnswer = questions[currentIndex].isTrueAnswer();
        intent.putExtra(KEY_EXTRA_ANSWER, correctAnswer);
        startActivity(intent);
});
```

20. W drugiej aktywności odczytaj przesyłaną wartość odpowiedzi na aktualne pytanie:

```
public class PromptActivity extends AppCompatActivity {

    private boolean correctAnswer;

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

        correctAnswer = getIntent().getBooleanExtra(MainActivity.KEY_EXTRA_ANSWER,  defaultValue: true);
    }
```

Metoda *getIntent()* zwraca zawsze ten obiekt intencji, który uruchomił daną aktywność (czyli był przekazany jako parametr metody *startActivity())*.
Pierwszy parametr metody do pobierania przesłanej informacji (*getBooleanExtra()*) to klucz, drugi zaś jest domyślną wartością, która zostanie zwrócona w przypadku, gdy podany klucz nie zostanie odnaleziony.

21. Odebrana odpowiedź powinna pojawiać się w drugiej aktywności po kliknięciu przycisku “Pokaż odpowiedź”:

```
showCorrectAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int answer = correctAnswer ? R.string.button_true : R.string.button_false;
        answerTextView.setText(answer);
    }
});
```

22. Uruchom aplikację i sprawdź poprawność działania.
23. Zmodyfikuj aplikację w ten sposób, aby istniała możliwość powrotu do pierwszej aktywności przesyłając jednocześnie informację o tym, że podpowiedź do danego pytania została wyświetlona. W tym celu w pierwszej aktywności (MainActivity) skorzystaj z metody *startActivityForResult()*:

```
promptButton.setOnClickListener((v) -> {
        Intent intent = new Intent( packageContext: MainActivity.this, PromptActivity.class);
        boolean correctAnswer = questions[currentIndex].isTrueAnswer();
        intent.putExtra(KEY_EXTRA_ANSWER, correctAnswer);
        startActivityForResult(intent, REQUEST_CODE_PROMPT);
});
```

gdzie:

```
private static final int REQUEST_CODE_PROMPT = 0;
```

Drugi parametr (stała REQUEST_CODE_PROMPT) oznacza numer aktywności wywołanej z poziomu danej aktywności, aby potem mieć możliwość odniesienia się do zwracanego wyniku – jest to szczególnie istotne w przypadku istnienia wielu wywołań różnych aktywności z poziomu danej aktywności.

24. W drugiej aktywności (PromptActivity) ustaw odpowiednio oczekiwany wynik, jako odpowiedź na kliknięcie przycisku “Pokaż odpowiedź”:

```
public class PromptActivity extends AppCompatActivity {
    public static final String KEY_EXTRA_ANSWER_SHOWN = "pb.edu.pl.wi.quiz.answerShown";
```

```
showCorrectAnswerButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int answer = correctAnswer ? R.string.button_true : R.string.button_false;
        answerTextView.setText(answer);
        setAnswerShownResult(true);
    }
});
private void setAnswerShownResult(boolean answerWasShown) {
    Intent resultIntent = new Intent();
    resultIntent.putExtra(KEY_EXTRA_ANSWER_SHOWN, answerWasShown);
    setResult(RESULT_OK, resultIntent);
}
```

25. Obsłuż odebranie wyniku w pierwszej aktywności, nadpisując metodę *onActivityResult()*:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (resultCode != RESULT_OK) { return; }
    if (requestCode == REQUEST_CODE_PROMPT) {
        if (data == null) { return; }
        answerWasShown = data.getBooleanExtra(PromptActivity.KEY_EXTRA_ANSWER_SHOWN, defaultValue: false);
    }
}
```

26. Ustaw domyślną wartość zmiennej *answerWasShown* na *false*:

```
nextButton.setOnClickListener((v) -> {
        currentIndex = (currentIndex + 1)%questions.length;
        answerWasShown = false;
        setNextQuestion();
});
```

27. Dodaj powiadomienie o wykryciu próby odpowiedzi na pytanie, do którego odpowiedź została już poznana:

```
<string name="answer_was_shown">Odpowiedź była już wyświetlona!</string>
```

```
private void checkAnswerCorrectness(boolean userAnswer) {
    boolean correctAnswer = questions[currentIndex].isTrueAnswer();
    int resultMessageId = 0;
    if (answerWasShown) {
        resultMessageId = R.string.answer_was_shown;
    } else {
        if (userAnswer == correctAnswer) {
            resultMessageId = "Prawidłowa odpowiedź";
        } else {
            resultMessageId = "Błąd! Nieprawidłowa odpowiedź";
        }
    }
    Toast.makeText( context: this, resultMessageId, Toast.LENGTH_SHORT).show();
}
```

28. Przetestuj działanie aplikacji.

PODSUMOWANIE:

- Na platformie CEZ należy zamieścić plik tekstowy, który zawiera:
    - Link do repozytorium github z wykonanym zadaniem
    - Wynik analizy cyklu życia aplikacji z wyjaśnieniem