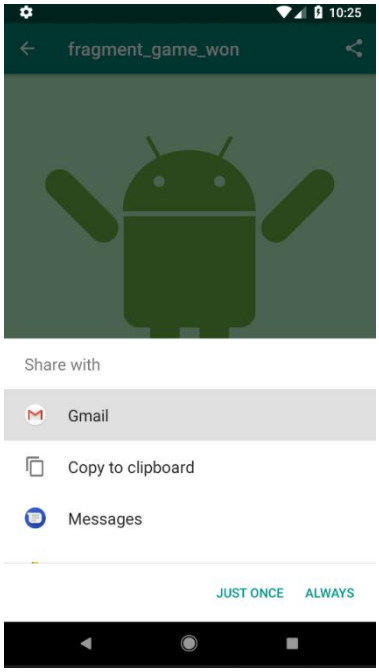
2. Intencje (Intent)

W tym ćwiczeniu zmodyfikujemy aplikację AndroidTrivia tak aby użytkownicy mogli wysyłać wyniki swoich gier do innych aplikacji i dzielić się nimi z przyjaciółmi.



3. Zadanie: skonfiguruj i używaj wtyczki Safe Args

Zanim użytkownicy będą mogli udostępniać wyniki swoich gier w aplikacji AndroidTrivia, kod musi przekazać parametry z jednego fragmentu do drugiego. Aby uniknąć błędów w tych transakcjach i uczynić je bezpiecznymi dla typu, korzystasz z wtyczki Gradle o nazwie *Safe Args*. Wtyczka generuje klasy `NavDirection` i dodajesz te klasy do swojego kodu.

W późniejszych zadaniach w tym ćwiczeniu używasz wygenerowanych klas `NavDirection` do przekazywania argumentów między fragmentami.

Dlaczego potrzebujesz wtyczki Safe Args

Często Twoja aplikacja będzie musiała przesyłać dane między fragmentami. Jednym ze sposobów przekazywania danych z jednego fragmentu do drugiego jest użycie instancji klasy `Bundle` class. Android `Bundle` to magazyn klucz-wartość.

Magazyn klucz-wartość, znany również jako słownik lub tablica asocjacyjna, to struktura danych, w której do pobrania wartości powiązanej z tym kluczem służy unikalny klucz (ciąg znaków). Na przykład:

Key	**Value**
"name"	"Anika"
"favorite_weather"	"sunny"
"favorite_color"	"blue"

Twoja aplikacja może użyć `Bundle` do przekazania danych z fragmentu A do fragmentu B. Na przykład fragment A tworzy bundle i zapisuje informacje jako pary klucz-wartość, a następnie przekazuje `Bundle` do fragmentu B. Następnie fragment B używa klucza do pobrania para klucz-wartość z `Bundle`. Ta technika działa, ale może, który się kompiluje ale może powodować błędy podczas działania aplikacji.

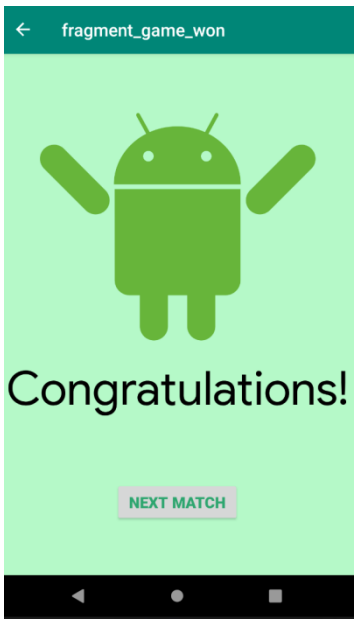
Rodzaje błędów, które mogą wystąpić:

- **Type mis-match errors**. Na przykład, jeśli fragment A wysyła ciąg, ale fragment B żąda liczby całkowitej z `Bundle`, żądanie zwraca domyślną wartość zero. Ponieważ zero jest prawidłową wartością, tego rodzaju problem z błędnym dopasowaniem typu nie generuje błędu podczas kompilacji aplikacji. Jednak gdy użytkownik uruchomi aplikację, błąd może spowodować nieprawidłowe działanie aplikacji lub awarię.
- **Missing key errors**. Jeśli fragment B żąda argumentu, który nie jest ustawiony w pakiecie, operacja zwraca wartość `null`. Ponownie, nie generuje to błędu podczas kompilacji aplikacji, ale może powodować poważne problemy, gdy użytkownik uruchomi aplikację.

Najlepiej wychwycić te błędy podczas kompilowania aplikacji w Android Studio, podczas tworzenia aplikacji, aby użytkownicy ich nie napotkali.

Aby rozwiązać te problemy, Android's Navigation Architecture Component includes zawiera funkcję *Safe Args*. Safe Args to wtyczka Gradle, która generuje kod i klasy, które pomagają wykrywać błędy w czasie kompilacji, które w innym przypadku mogłyby nie zostać ujawnione, dopóki aplikacja nie zostanie uruchomiona.

Krok 1: Otwórz aplikację startową i uruchom ją



W tym ćwiczeniu dodajesz ikonę udostępniania na górze ekranu Gratulacje. **share icon** umożliwia użytkownikowi udostępnianie wyników w wiadomości e-mail lub tekście.

Krok 2: Dodaj bezpieczne argumenty do projektu

1. W Android Studio otwórz plik build.gradle na poziomie projektu.
2. Dodaj zależność `navigation-safe-args-gradle-plugin` jak pokazano poniżej:

```
// Adding the safe-args dependency to the project Gradle file
dependencies {
   ...
classpath "android.arch.navigation:navigation-safe-args-gradle-
plugin:$navigationVersion"

}
```

3. Otwórz plik build.gradle na poziomie aplikacji.
4. W górnej części pliku, po wszystkich innych wtyczkach, dodaj instrukcję `apply plugin` wraz z wtyczką `androidx.navigation.safeargs`:

```
// Adding the apply plugin statement for safeargs
apply plugin: 'androidx.navigation.safeargs'
```

5. Przebuduj projekt. Jeśli pojawi się monit o zainstalowanie dodatkowych narzędzi do kompilacji, zainstaluj je.

Projekt aplikacji zawiera teraz wygenerowane klasy `NavDirection` .

Wtyczka Safe Args plugin generuje klasę `NavDirection` dla każdego fragmentu. Klasy reprezentują nawigację ze wszystkich działań aplikacji.

Na przykład, `GameFragment` ma teraz wygenerowaną klasę `GameFragmentDirections` Za pomocą klasy `GameFragmentDirections` można przekazywać bezpieczne dla typu argumenty między fragmentem gry a innymi fragmentami w aplikacji.

Aby zobaczyć wygenerowane pliki, przejrzyj folder **generatedJava** w panelu **Project** > **Android**.

Uwaga!: Nie edytuj klas NavDirection. Klasy te są generowane za każdym razem, gdy projekt jest kompilowany, a wprowadzone zmiany zostaną utracone.

Krok 3: Dodaj klasę NavDirection do fragmentu gry

W tym kroku dodajesz klasę `GameFragmentDirections` do fragmentu gry. Później użyjesz tego kodu do przekazania argumentów między `GameFragment` a fragmentami stanu gry (`GameWonFragment` and `GameOverFragment`).

1. Otwórz plik `GameFragment.kt` , który znajduje się w folderze Java.
2. W metodzie `onCreateView()` zlokalizuj wyrażenie warunkowe ("We've won!"). Zmień parametr przekazywany do metody `NavController.navigate()` Zamień identyfikator akcji dla stanu wygranej na identyfikator, który używa metoda `actionGameFragmentToGameWonFragment()` z klasy `GameFragmentDirections` .

Instrukcja warunkowa wygląda teraz jak poniższy kod. Dodacie parametry do metody `actionGameFragmentToGameWonFragment()` w następnym zadaniu.

```
// Using directions to navigate to the GameWonFragment
view.findNavController()

.navigate(GameFragmentDirections.actionGameFragmentToGameWonFragment())
```

3. Podobnie znajdź instrukcję warunkową po zakończeniu gry ("Game over!"). Zamień identyfikator akcji dla stanu Game Over na ID, który używa metody Game-Over z klasy `GameFragmentDirections`:

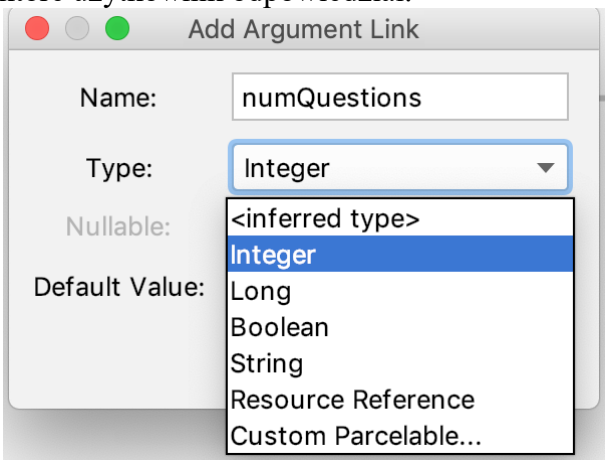
```
// Using directions to navigate to the GameOverFragment
view.findNavController()

.navigate(GameFragmentDirections.actionGameFragmentToGameOverFragment())
```

4. Zadanie: dodaj i przekaż argumenty

W tym zadaniu dodajesz wpisane argumenty do `gameWonFragment` i przekazujesz argumenty do metody `GameFragmentDirections` Następnie zastępujesz pozostałe klasy fragmentów ich równoważnymi klasami `NavDirection`.

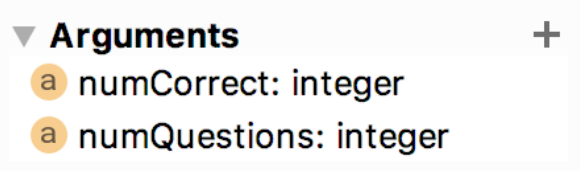
Krok 1: Dodaj argumenty do fragmentu wygranej (game-won fragment)

1. Otwórz plik `navigation.xml` który znajduje się w folderze **res** > **navigation** Kliknij kartę **Design** aby otworzyć graf nawigacyjny (navigation graph), w którym ustawisz argumenty we fragmentach.
2. W podglądzie wybierz **gameWonFragment**.
3. W panelu **Attributes** rozwiń sekcję **Arguments**.
4. Kliknij ikonę + icon aby dodać argument. Nazwij argument **numQuestions** i ustaw typ na Integer, a następnie kliknij **Add**. Ten argument reprezentuje liczbę pytań, na które użytkownik odpowiedział.



5. Nadal przy wybranym **gameWonFragment** dodaj drugi argument. Nazwij ten argument **numCorrect** i ustaw jego typ na **Integer**. en argument reprezentuje liczbę

pytań, na które użytkownik poprawnie odpowiedział.



Jeśli spróbujesz teraz zbudować aplikację, prawdopodobnie wystąpią dwa błędy kompilacji.

```
No value passed for parameter 'numQuestions'
No value passed for parameter 'numCorrect'
```

Naprawiasz ten błąd w kolejnych krokach.

Uwaga: jeśli używasz Androida Studio 3.2 lub starszego, może być konieczna zmiana `app:type = "integer"` na `app:argType = "integer"` w pliku `navigation.xml`.

Krok 2: Przekaż argumenty

W tym kroku przekazujesz argumenty `numQuestions` i `questionIndex` do metody `actionGameFragmentToGameWonFragment()` z klasy `GameFragmentDirections`.

1. Otwórz plik `GameFragment.kt` zlokalizuj `We've won`:

```
else {
 // We've won!  Navigate to the gameWonFragment.
 view.findNavController()
      .navigate(GameFragmentDirections
            .actionGameFragmentToGameWonFragment())
}
```

2. Przekaż parametry `numQuestions` i `questionIndex` do metody `actionGameFragmentToGameWonFragment()`:

```
// Adding the parameters to the Action
view.findNavController()
      .navigate(GameFragmentDirections
            .actionGameFragmentToGameWonFragment(numQuestions,
questionIndex))
```

Podajesz całkowitą liczbę pytań jako `numQuestions` a bieżące pytanie jako `questionIndex`. Aplikacja została zaprojektowana w taki sposób, że użytkownik może udostępniać swoje dane tylko wtedy, gdy odpowie poprawnie na wszystkie pytania - liczba poprawnych pytań zawsze odpowiada liczbie pytań, na które udzielono odpowiedzi. (Możesz zmienić tę logikę gry później, jeśli chcesz).

3. W `GameWonFragment.kt`, extract wyodrębnij argumenty z pakietu, a następnie użyj `Toast` aby wyświetlić argumenty. Umieść następujący kod w metodzie `onCreateView()` przed instrukcją `return` :

```
val args = GameWonFragmentArgs.fromBundle(arguments!!)
```

```
Toast.makeText(context, "NumCorrect: ${args.numCorrect}, NumQuestions:
${args.numQuestions}", Toast.LENGTH_LONG).show()
```

4. Uruchom aplikację i zagraj w grę, aby upewnić się, że argumenty zostały pomyślnie przekazane do `GameWonFragment`. Komunikat toast pojawia się na ekranie **Congratulations pokazując** "NumCorrect: 3, NumQuestions: 3".

   Aby ułatwić grę, możesz zmienić ją na grę z jednym pytaniem, ustawiając wartość `numQuestions` na `1` w pliku `GameFragment.kt`.

Krok 3: Zamień klasy fragmentów na klasy NavDirection

Kiedy używasz „bezpiecznych argumentów”, możesz zastąpić klasy fragmentów używane w kodzie nawigacji klasami `NavDirection` Robisz to, abyś mógł używać argumentów bezpiecznych dla typu z innymi fragmentami w aplikacji.

W `TitleFragment`, `GameOverFragment`, i `GameWonFragment`, zmień identyfikator akcji przekazywany do metody `navigate()` Zastąp identyfikator akcji równoważną metodą z odpowiedniej klasy `NavDirection`:

1. Otwórz plik `TitleFragment.kt` w metodzie `onCreateView()`,zlokalizuj metodę `navigate()` w module obsługi kliknięcia przycisku**Play** Przekaż `TitleFragmentDirections.actionTitleFragmentToGameFragment()` jako argument metody:

```
binding.playButton.setOnClickListener { view: View ->
    view.findNavController()

.navigate(TitleFragmentDirections.actionTitleFragmentToGameFragment())
}
```

2. W pliku `GameOverFragment.kt` w module obsługi kliknięć przycisku **Try Again**, przekaż `GameOverFragmentDirections.actionGameOverFragmentToGameFragment()` as jako argument metody `navigate()`:

```
binding.tryAgainButton.setOnClickListener { view: View ->
    view.findNavController()

.navigate(GameOverFragmentDirections.actionGameOverFragmentToGameFragment()
)
}
```

3. W pliku `GameWonFragment.kt` w module obsługi kliknięć przycisku **Next Match** przekaż `GameWonFragmentDirections.actionGameWonFragmentToGameFragment()` ako argument metody `navigate()`:

```
binding.nextMatchButton.setOnClickListener { view: View ->
    view.findNavController()

.navigate(GameWonFragmentDirections.actionGameWonFragmentToGameFragment())
```

```
}
```

4. Uruchom aplikację.

   Nie znajdziesz żadnych zmian w danych wyjściowych aplikacji, ale teraz aplikacja jest skonfigurowana tak, że możesz łatwo przekazywać argumenty za pomocą klas `NavDirection` w razie potrzeby.

5. Zadanie: dodaj „implicit intent" i element menu „udostępnij"

W tym zadaniu używasz implicit intent aby dodać funkcję udostępniania do aplikacji, aby użytkownik mógł udostępniać swoje wyniki gry. Implementujesz funkcję udostępniania jako menu opcji w klasie `GameWonFragment` W interfejsie aplikacji element menu pojawi się jako ikona udostępniania u góry ekranu **Congratulations**.

Implicit intents (Domniemane intencje)

Do tej pory korzystałeś z komponentów nawigacyjnych, aby poruszać się między fragmentami swojej aktywności. Android pozwala także na korzystanie z *intents* w celu nawigowania do activities które zapewniają inne aplikacje. Korzystasz z tej funkcji w aplikacji AndroidTrivia, aby umożliwić użytkownikowi udostępnianie wyników gry.

Intent to prosty obiekt wiadomości służący do komunikacji między komponentami Androida. Z *implicit intent*, inicjujesz activity nie wiedząc, która aplikacja lub activity obsłuży to zadanie. Na przykład, jeśli chcesz, aby aplikacja zrobiła zdjęcie, zazwyczaj nie obchodzi Cię, która aplikacja lub activity wykonuje to zadanie. Gdy wiele aplikacji na Androida może obsłużyć ten sam *implicit intent* (dorozumiany zamiar), system Android pokazuje użytkownikowi wybór, dzięki czemu użytkownik może wybrać aplikację do obsługi żądania.

Każda implicit intent musi mieć `ACTION` które opisuje rodzaj rzeczy do zrobienia. Typowe akcje, takie jak `ACTION_VIEW`, `ACTION_EDIT`, i `ACTION_DIAL`, są zdefiniowane w klasie `Intent`.

Alert terminologiczny! `Intent` actions nie są powiązane z actions pokazanymi na grafie nawigacyjnym.

Krok 1: Dodaj menu opcji do ekranu Gratulacje Congratulations screen

1. Otwórz plik `GameWonFragment.kt`.
2. Wewnątrz metody `onCreateView()` przed `return`, wywołaj metodę `setHasOptionsMenu()` i przekaż `true`:

```
setHasOptionsMenu(true)
```

Krok 2: Zbuduj i wywołaj implicit intent

Zmodyfikuj kod, aby skompilować i wywołać `Intent` który wyśle wiadomość o danych gry użytkownika. Ponieważ kilka różnych aplikacji może obsłużyć `ACTION_SEND` intent, użytkownik zobaczy selektor, który pozwoli mu wybrać sposób wysłania informacji.

1. W klasie `GameWonFragment` po metodzie `onCreateView()` method utwórz prywatną metodę o nazwie `getShareIntent()`,jak pokazano poniżej. Wiersz kodu ustawiający wartość argumentów jest identyczny z wierszem kodu używanym w klasie `onCreateView()`.

   W pozostałej części kodu metody budujesz `ACTION_SEND` intent z zamiarem dostarczenia wiadomości, którą użytkownik chce udostępnić. Typ MIME danych określa metoda `setType()` Rzeczywiste dane, które należy dostarczyć, są określone w `EXTRA_TEXT`. (`share_success_text` jest zdefiniowany w pliku zasobów `strings.xml`.)

```
// Creating our Share Intent
private fun getShareIntent() : Intent {
   val args = GameWonFragmentArgs.fromBundle(arguments!!)
   val shareIntent = Intent(Intent.ACTION_SEND)
        shareIntent.setType("text/plain")
            .putExtra(Intent.EXTRA_TEXT,
getString(R.string.share_success_text, args.numCorrect, args.numQuestions))
   return shareIntent
}
```

2. Poniżej metody `getShareIntent()` utwórz metodę `shareSuccess()` Ta metoda pobiera `Intent` z `getShareIntent()` i wywołuje `startActivity()` w celu rozpoczęcia udostępniania.

```
// Starting an Activity with our new Intent
private fun shareSuccess() {
   startActivity(getShareIntent())
}
```

3. Kod startowy zawiera już plik menu `winner_menu.xml` Zastąp funkcję `onCreateOptionsMenu()` aby nadmuchać menu `winner_menu`.

   Użyj `getShareIntent()` aby uzyskać `shareIntent`. Aby upewnić się, że `shareIntent` widzi `Activity`, sprawdź w menedżerze pakietów Androida (`PackageManager`), który śledzi aplikacje i działania zainstalowane na urządzeniu. Użyj właściwości `packageManager` aby uzyskać dostęp do menedżera pakietów i wywołać funkcję `resolveActivity()` .Jeśli wynik jest równy null, co oznacza, że `null`, co oznacza, że `shareIntent` nie zostanie rozwiązany, znajdź pozycję menu udostępniania z napompowanego menu i spraw, aby pozycja menu była niewidoczna..

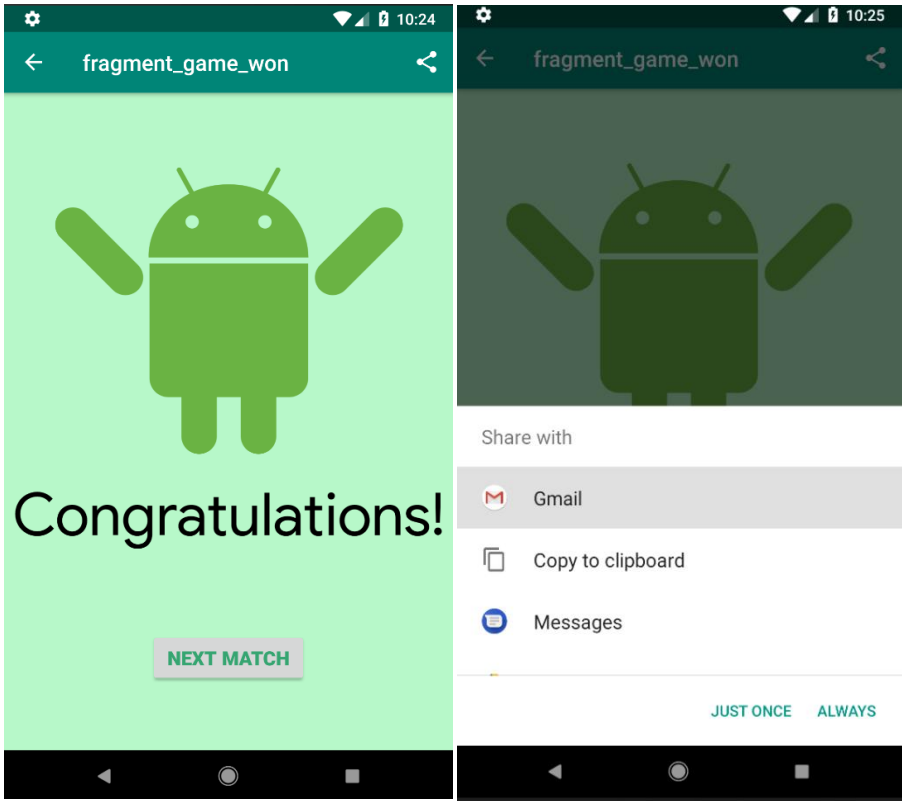
```
// Showing the Share Menu Item Dynamically
override fun onCreateOptionsMenu(menu: Menu?, inflater: MenuInflater?) {
   super.onCreateOptionsMenu(menu, inflater)
   inflater?.inflate(R.menu.winner_menu, menu)
   // check if the activity resolves
```

```
    if (null == getShareIntent().resolveActivity(activity!!.packageManager))
{
        // hide the menu item if it doesn't resolve
        menu?.findItem(R.id.share)?.setVisible(false)
    }
}
```

4. Aby obsłużyć pozycję menu, przesłoń `onOptionsItemSelected()`.Wywołaj metodę `shareSuccess()` po kliknięciu elementu menu:

```
// Sharing from the Menu
override fun onOptionsItemSelected(item: MenuItem?): Boolean {
    when (item!!.itemId) {
        R.id.share -> shareSuccess()
    }
    return super.onOptionsItemSelected(item)
}
```

5. Teraz uruchom aplikację. (Może być konieczne zaimportowanie niektórych pakietów do `GameWonFragment.kt` before the code will run.) przed uruchomieniem kodu.) Po wygraniu gry zwróć uwagę na ikonę udostępniania, która pojawia się w prawym górnym rogu paska aplikacji. Kliknij ikonę udostępniania, aby udostępnić wiadomość o swoim zwycięstwie.



7. 7. Podsumowanie

Safe Args(Bezpieczne argumenty):

- Aby uchwycić błędy spowodowane brakującymi kluczami lub błędnie dopasowanymi typami podczas przesyłania danych z jednego fragmentu do drugiego, użyj wtyczki Gradle o nazwie *Safe Args*.
- Dla każdego fragmentu w aplikacji wtyczka Safe Args generuje odpowiednią klasę `NavDirection` Dodajesz klasę `NavDirection` do kodu fragmentu, a następnie używasz tej klasy do przekazywania argumentów między fragmentem a innymi fragmentami.
- Klasy `NavDirection` reprezentują nawigację ze wszystkich działań aplikacji.

Implicit intents:

- *implicit intent* Domniemany zamiar deklaruje działanie, które twoja aplikacja chce, aby inna aplikacja (na przykład aplikacja aparatu fotograficznego lub aplikacja e-mail) wykonała w jej imieniu.
- Jeśli kilka aplikacji na Androida może obsłużyć implicit intent, Android pokazuje użytkownikowi wybór. Na przykład, gdy użytkownik stuknie ikonę udostępniania w aplikacji AndroidTrivia, może wybrać aplikację, której chce użyć, aby udostępnić wyniki swojej gry.
- Aby zbudować intent, deklarujesz akcję do wykonania, na przykład `ACTION_SEND`.
- Dostępnych jest kilka konstruktorów `Intent()` które pomagają w budowaniu intents.

Sharing functionality (Funkcja udostępniania):

- W przypadku dzielenia się sukcesem ze znajomymi, zamierzoną czynnością byłoby `Intent.ACTION_SEND`.
- Aby dodać menu opcji do fragmentu, ustaw metodę `setHasOptionsMenu()` na `true` w kodzie fragmentu..
- W kodzie fragmentu zastąp metodę `onCreateOptionsMenu()` aby napompować menu.
- Zastąp `onOptionsItemSelected()` aby użyć`startActivity()` w celu wysłania `Intent` do innych aplikacji, które mogą go obsłużyć.