Тема 8. Життєві цикли візуальних компонентів

8.1 Операції (Activity)

Activity – це компонент програми, який видає екран і з яким користувачі можуть взаємодіяти, щоб виконати будь-які дії, наприклад набрати номер телефону, зробити фото, відправити лист або переглянути карту. Кожній операції присвоюється вікно для промальовування відповідного, призначеного для користувача інтерфейсу. Зазвичай вікно відображається на весь екран, проте його розмір може бути меншим, і воно може розміщуватися поверх інших вікон.

Як правило, програма містить кілька операцій, які слабо пов’язані одна з одною. Зазвичай одна з операцій в додатку позначається як «основна», пропонована користувачеві при першому запуску програми. У свою чергу, кожна операція може запустити іншу операцію для виконання різних дій. Кожен раз, коли запускається нова операція, попередня операція зупиняється, однак система зберігає її в стек («стек переходів назад»). При запуску нової операції вона поміщається в стек переходів назад і відображається для користувача. Стек переходів назад працює за принципом «останнім увійшов – першим вийшов», тому після того як користувач завершив поточну операцію і натиснув кнопку Назад, поточна операція видаляється зі стека (і знищується), і відновлюється попередня операція.

Коли операція зупиняється через запуск нової операції, для повідомлення про зміну її стану використовуються методи зворотного виклику життєвого циклу операції. Існує кілька таких методів, які може приймати операція внаслідок зміни свого стану: створення операції, її зупинка, відновлення або знищення системою; також кожен зворотний виклик дає можливість виконати певну дію, відповідну для певної зміни стану. Наприклад, в разі зупинки операція повинна звільнити будь-які великі об’єкти, наприклад, підключення до мережі або базі даних. При відновленні операції можна повторно отримати необхідні ресурси і відновити виконання перерваних дій. Такі зміни стану є частиною життєвого циклу операції.

Далі розглянемо основи створення і використання операцій, включаючи повний опис життєвого циклу операції, щоб краще зрозуміти, як слід керувати переходами між різними станами операції.

Створення операції. Щоб створити операцію, спочатку необхідно створити підклас класу Activity (або скористатися існуючим його підкласом). У такому підкласі необхідно реалізувати методи зворотного виклику, які викликає система при переході операції з одного стану свого життєвого циклу в інший, наприклад, при створенні, зупинці, відновленні або знищенні операції. Ось два найбільш важливих методи зворотного виклику:

– onCreate(). Цей метод необхідно обов'язково реалізувати, оскільки система викликає його при створенні вашої операції. У своїй реалізації вам необхідно ініціалізувати ключові компоненти операції. Найбільш важливо саме тут викликати setContentView() для визначення макета призначеного для користувача інтерфейсу операції.
– onPause(). Система викликає цей метод як першу ознаку виходу користувача з операції (однак це не завжди означає, що операція буде знищена). Зазвичай саме тут необхідно застосовувати будь-які зміни, які повинні бути збережені, крім поточного сеансу роботи користувача (оскільки користувач може не повернутися назад).

Існують також і деякі інші методи зворотного виклику життєвого циклу, які необхідно використовувати для того, щоб забезпечити плавний перехід між операціями, а також для обробки непередбачених порушень у роботі операції, в результаті яких вона може бути зупинена або навіть знищена.

Реалізація інтерфейсу користувача. Для реалізації призначеної для інтерфейсу користувача операції використовується ієрархія подань-об'єктів, отриманих з класу View. Кожне подання відповідає за певну прямокутну область вікна операції і може реагувати на дії користувачів. Наприклад, поданням може бути кнопка, натискання на яку приводить до виконання певної дії.

В Android передбачений набір вже готових подань, які можна використовувати для створення дизайну макета і його організації.

Віджети – це подання з візуальними (і інтерактивними) елементами, наприклад кнопками, текстовими полями, чекбоксами або просто зображеннями.

Макети – це подання, отримані з класу ViewGroup, що забезпечують унікальну модель компонування для своїх дочірніх подань, таких, як лінійний макет, сітка або відносний макет. Також можна створити підклас для класів View та ViewGroup (або скористатися існуючими підкласами), щоб створити власні віджети і макети, а потім застосувати їх до макета своєї операції.

Найчастіше для задання макета за допомогою подань використовується XML-файл макета, збережений в ресурсах додатка. Таким чином можна зберігати дизайн інтерфейсу користувача окремо від вихідного коду, який служить для задання поведінки операції. Щоб задати макет, призначений для користувацького інтерфейсу, можна використовувати метод setContentView(), передавши в нього ідентифікатор ресурсу для макета. Однак можна створити нові View в коді вашої операції і створити ієрархію подань. Для цього вставте View в ViewGroup, а потім використовуйте цей макет, передавши кореневий об’єкт ViewGroup в метод setContentView().

При створенні нової програми за допомогою інструментів Android SDK в заготовці операції, створюваної автоматично, є фільтр намірів, який оголошує операцію. Ця операція реагує на виконання «основної» дії, і її слід помістити в категорію «перехід засобу запуску».

Якщо додаток планується створити самодостатнім і заборонити іншим додаткам активувати його операції, то інших фільтрів намірів створювати не потрібно. У цьому випадку тільки в одній операції має бути «основна» дія, і її слід помістити в категорію засобу запуску, як в прикладі, наведеному вище. В операції, які не повинні бути доступні для інших додатків, не слід включати фільтри намірів. Такі операції можна самостійно запустити за допомогою явних намірів.

Однак, якщо треба, щоб операція реагувала на неявні наміри, одержувані від інших додатків (а також з вашого додатка), для операції слід визначити додаткові фільтри намірів. Для кожного типу наміру, на який необхідно

реагувати, потрібно вказати об’єкт <intent-filter>, що включає елемент <action> і необов’язковий елемент <category> чи <data> (або обидва ці елементи). Ці елементи визначають тип наміру, на який може реагувати ваша операція.

Запуск операції. Для запуску іншої операції досить викликати метод startActivity(), передавши в нього об’єкт Intent, який описує операцію, що запускається. У намірі або вказується точна операція для запуску, або описується тип операції, яку ви хочете виконати (після цього система вибирає для вас підходящу операцію, яка може навіть перебувати в іншому додатку). Намір також може містити невеликий обсяг даних, які будуть використовуватися запущеною операцією.

При роботі з власним додатком найчастіше треба лише запустити потрібну операцію. Для цього необхідно створити намір, який явно визначає необхідну операцію з допомогою імені класу. Нижче наведено приклад запуску однією операцією іншої операції з ім’ям SignInActivity:

Однак у вашому додатку також може стати потрібним виконати деяку дію, наприклад, відправити лист, текстове повідомлення або оновити статус, використовуючи дані з вашої операції. В цьому випадку у вашому додатку можуть бути відсутні такі дії, тому ви можете скористатися операціями з інших додатків, наявних на пристрої, які можуть виконувати необхідні дії. Саме в цьому разі наміри особливо корисні – можна створити намір, який описує необхідну дію, після чого система запускає його з іншої програми. За наявності декількох операцій, які можуть обробити намір, користувач може вибирати, який з них слід використовувати. Наприклад, якщо користувачеві потрібно надати можливість відправити електронний лист, можна створити такий намір:

Завершення операції. Для завершення операції досить викликати її finish(). Також для завершення окремої операції, запущеної раніше, можна викликати метод finishActivity().

Управління життєвим циклом операцій

Управління життєвим циклом операцій шляхом реалізації методів зворотного виклику має важливе значення при розробці надійних і гнучких програм. На життєвий цикл операції безпосередньо впливають його зв'язки з іншими операціями, його завданнями та стком переходів назад.

Існує всього три стани операції:

- Відновлена – операція виконується на передньому плані екрана і відображається для користувача (цей стан операції також іноді називається «Виконувана»).
- Припинена – на передньому фоні виконується інша операція, яка відображається для користувача, однак ця операція, як і раніше, не прихована. Тобто поверх поточної операції показується інша операція, частково прозора або така, що не займає повністю весь екран. Призупинена операція повністю активна (об'єкт Activity, як і раніше, знаходиться в пам'яті, в ньому зберігаються всі відомості про стан і інформація про елементи, і він також залишається пов'язаним з диспетчером вікон), проте в разі гострого браку пам'яті система може завершити її.
- Зупинена – операція повністю перекривається іншою операцією (тепер вона виконується у фоновому режимі). Зупинена операція, як і раніше, активна (об'єкт Activity, як і раніше, знаходиться в пам'яті, в ньому зберігаються всі відомості про стан і інформація про елементи, але об'єкт більше не пов'язаний з диспетчером вікон). Однак операцію більше не видно користувачеві, і в разі нестачі пам'яті система може завершити її.

Якщо операція припинена або повністю зупинена, система може очистити її з пам'яті, завершивши саму операцію (за допомогою методу finish()), або просто завершити її процес. У разі повторного відкриття операції (після її завершення) її потрібно створити повністю.

Реалізація зворотних викликів життєвого циклу. При переході операції з одного, описаного вище стану в інший, повідомлення про це реалізуються через різні методи зворотного виклику. Всі методи зворотного виклику є прив'язками, які можна перевизначити для виконання відповідної дії при зміні стану операції. Зазначена нижче базова операція включає кожен з основних методів життєвого циклу:

Примітка. При реалізації методів життєвого циклу завжди викликайте реалізацію суперкласу, перш ніж виконувати будь-які дії, як показано в прикладах вище.

Разом всі ці методи визначають весь життєвий цикл операції. За допомогою реалізації цих методів можна відстежувати три вкладених цикли в життєвому циклі операції:

1. Весь життєвий цикл операції – відбувається між викликом методу onCreate() і викликом методу onDestroy(). Ваша операція повинна виконати настройку «глобального» стану (наприклад, визначення макета) в методі onCreate(), а потім звільнити всі ресурси, що залишилися в onDestroy(). Наприклад, якщо у вашій операції є потік, що виконується у фоновому режимі, для завантаження даних по мережі, операція може створити такий потік в методі onCreate(), а потім зупинити його в методі onDestroy().

2. Видимий життєвий цикл операції – відбувається між викликами методів onStart() та onStop(). Протягом цього часу операція відображається на екрані, де користувач може взаємодіяти з нею. Наприклад, метод onStop() викликається в разі, коли запускається нова операція, а поточна більше не відображається. У проміжку між викликами цих двох методів можна зберегти ресурси, необхідні для відображення операції для користувача. Наприклад, можна зареєструвати об'єкт BroadcastReceiver в методі onStart() для відстеження змін, що впливають на призначений для користувача інтерфейс, а потім скасувати його реєстрацію в методі onStop(), коли користувач більше не бачить відображуваного. Протягом усього життєвого циклу операції система може кілька разів викликати методи

onStart() та onStop(), оскільки операція то відображається для користувача, то ховається від нього.

3. Життєвий цикл операції, що виконується на передньому плані – відбувається між викликами методів onResume() та onPause(). Протягом цього часу операція виконується на тлі всіх інших операцій і відображається для користувача. Операція може часто входити у фоновий режим і виходити з нього. Наприклад, метод onPause() викликається при переході пристрою в сплячий режим або при появі діалогового вікна. Оскільки перехід у такий стан може виконуватися досить часто, код в цих двох методах повинен бути легким, щоб не допустити повільних переходів і не змушувати користувача чекати.

Збереження стану операції

В оглядових відомостях про управління життєвим циклом операції коротко згадується, що в разі припинення або повної зупинки операції її стан зберігається. Це дійсно так, оскільки об’єкт Activity при цьому, як і раніше, знаходиться в пам’яті, і вся інформація про її елементи і поточний стан, як і раніше, активна. Тому будь-які внесені користувачем в операції зміни зберігаються, і коли операція повертається на передній план (коли вона «відновлюється»), ці зміни залишаються в цьому об’єкті.

Однак коли система знищує операцію з метою відновлення пам’яті, об’єкт Activity знищується, в результаті чого системі не вдається просто відновити стан операції для взаємодії з нею. Замість цього системі необхідно повторно створити об’єкт Activity, якщо користувач повертається до нього. Але користувачеві невідомо, що система вже знищила операцію і створила її повторно, а тому, можливо, він очікує, що операція залишилася колишньою. У цій ситуації можна забезпечити збереження важливої інформації про стан операції шляхом реалізації додаткового методу зворотного виклику, який дозволяє зберегти інформацію про ваші операції: onSaveInstanceState().

Перш ніж зробити операцію доступною для знищення, система викликає метод onSaveInstanceState(). Вона передає в цей метод об’єкт Bundle, в якому

можна зберегти інформацію про стан операції у вигляді пари «ім’я- значення», використовуючи для цього такі методи, як putString() та putInt(). Потім, якщо система завершує процес вашого застосування і користувач повертається до вашої операції, система повторно створює операцію і передає об’єкт Bundle в обидва методи: onCreate() та onRestoreInstanceState(). За допомогою будь-якого з цих методів можна витягти з об’єкта Bundle збережену інформацію про стан операції і відновити її. Якщо така інформація відсутня, то об’єкт Bundle передається з нульовим значенням (це відбувається в разі, коли операція створюється в перший раз).

Є два способи повернення операції до відображення для користувача в незміненому стані: знищення операції з подальшим її повторним створенням, коли операція повинна відновити свій раніше збережений стан, або зупинка операції та її подальше відновлення в незміненому стані.

Відмінний спосіб перевірити спроможність вашого додатку відновлювати свій стан – це просто повернути пристрій для зміни орієнтації екрана. При зміні орієнтації екрана система знищує і повторно створює операцію, щоб застосувати альтернативні ресурси, які можуть бути доступні для нової конфігурації екрана. Тільки з однієї цієї причини вкрай важливо, щоб ваша операція могла повністю відновлювати свої стани при її повторному створенні, оскільки користувачі постійно працюють з додатками в різних орієнтаціях екрана.

Обробка змін в конфігурації

Деякі конфігурації пристроїв можуть змінюватися в режимі виконання (наприклад, орієнтація екрана, доступність клавіатури і мова). У таких випадках Android повторно створює виконання повсякденних завдань (система спочатку викликає метод onDestroy(), а потім відразу ж викликає метод onCreate()). Така поведінка дозволяє додатку враховувати нові конфігурації шляхом автоматичного перезавантаження в додаток альтернативних ресурсів, які були надані (наприклад, різні макети для різних орієнтацій і екранів різних розмірів).

Якщо операція розроблена належним чином і належним чином підтримує перезапуск після зміни орієнтації екрана і відновлення свого стану, як описано вище, ваш додаток можна вважати більш стійким до інших непередбачених подій в життєвому циклі операції.

Кращий спосіб обробки такого перезапуску – зберегти і відновити стан операції за допомогою методів onSaveInstanceState() та onRestoreInstanceState() (чи onCreate()).

Узгодження операцій

Коли одна операція запускає іншу, в життєвих циклах обох з них відбувається перехід з одного стану в інший. Перша операція припиняється і завершується (проте вона не буде зупинена, якщо вона, як і раніше, видима на фоні), а друга операція створюється. У разі, якщо ці операції обмінюються даними, збереженими на диску або в іншому місці, важливо розуміти, що перша операція не зупиняється повністю доти, поки не буде створена друга операція. Навпаки, процес запуску другої операції накладається на процес зупинки першої операції.

Порядок зворотних викликів життєвого циклу чітко визначено, зокрема, коли в одному і тому ж процесі знаходяться дві операції, і одна з них запускає іншу. Наведемо порядок виконання дій у разі, коли операція А запускає операцію Б:

1. Виконується метод onPause() операції А. Послідовно виконуються методи onCreate(), onStart() та onResume() операції Б (тепер для користувача відображається операція Б).
2. Потім, якщо операція А більше не відображається на екрані, виконується її метод onStop().

Така передбачувана послідовність виконання зворотних викликів життєвого циклу дозволяє управляти переходом інформації з однієї операції в іншу. Наприклад, якщо після зупинки першої операції потрібно виконати запис в базу даних, щоб наступна операція могла зчитувати їх, то запис в базу даних

слід виконати під час виконання методу onPause(), а не під час виконання методу onStop().