# Razvoj Android aplikacija

# Aktivnosti

U razvoju aplikacija koriste se komponente koje su povezane u AndroidManifest.xml datoteci. Ova datoteka opisuje svaku komponentu aplikacije i njihovo međusobno djelovanje. Glavne komponente koje se koriste u Android aplikaciji su: aktivnosti i servisi.

**Aktivnosti** – (*activities*) određuju korisničko sučelje i upravljaju korisničkim aktivnostima na zaslonu uređaja. Jedna aktivnost predstavlja jedan zaslon s korisničkim sučeljem koji vidi korisnik. Ova komponenta izvršava aktivnosti na ekranu.

**Primjer**: email aplikacija može imati jednu aktivnost koja se odnosi na prikaz liste novih poruka, dok se druga aktivnost odnosi na čitanje poruka.

Aktivnost se implementira kao podklasa AppCompatActivity klase:

**public class** MainActivity **extends** AppCompatActivity{}

1. **Servisi – (***services)*

Servisi upravljaju pozadinskim procesima koji su vezani s aplikacijom koja obavlja neke rutinske operacije u određenim intervalima.

**Primjer**: servis komponenta može obavljati dugotrajnu operaciju prijenosa podataka preko mreže, a da se pri tome ne zaustavlja korisničko izvršenje akcije.

Servis se implementira kao podklasa Service klase:

**public class** MyService **extends** Service {}

1. Servisi

Servis izvršava dugotrajne i zahtjevne operacije u pozadini za koje ne postoji potreba da ih aktivira korisnik tako da za njih nema neko korisničko sučelje. Druga komponenta aplikacije može pokrenuti servis i on će se nastaviti izvršavati u pozadini i ako korisnik prijeđe na neku drugu aplikaciju.

Komponenta se može povezati sa servisom i integrirati s njim i tako omogućiti njihovu komunikaciju (interprocess comunication, IPC)

**Primjer**: servis može upravljati transakcijama preko mreže, puštati muziku, izvršiti I/O operacije ili komunicirati sa content providerom i to sve u pozadini.

Servis se pojavljuje u dvije forme:

1. Pokrenuti servis(Started) – kada ga komponenta aplikacije pozivom metode startService() pokrene. Jednom pokrenut servis može se izvršavati u pozadini neograničeno dugo, bezobzira na trenutni status komponente. Pokrenuti servisi najčešće ne vraćaju neki rezultat komponenti koja ga poziva, već samo izvrše zatraženi zadatak i zaustave se.

Primjer: download/upload datoteke preko mreže.

1. Povezani(Bound) – kada se komponenta aplikacije vezuje sa sevisom pomoću metode bindService(). Tada se servis izvršava dok je aktivna komponenta ili komponente koje su s njim povezane. Vezani servisi kroz IPC omogućavaju klijent-server sučelje, slanje zahtjeva, vraćanje rezultata, i sl. Ovakvi servisi rade sve dok su vezani za neku komponentu i moguće je vezati ih za više komponenti istovremeno. Kada se u potpunosti odvežu od komponente servisi se ukidaju.

Bez obzira na tip servisa, komponente aplikacije mogu korstiti servis isto kao i aktivnost–pokrenuti ga pomoću intent-a.

Servis u okviru Android Software Development Kit (SDK) označava dvije stvari:

1. ili je proces u pozadini, koji obavlja neke korisne informacije u redovnim intervalima.
2. ili je sučelje za udaljeni objekt koji se zove u okviru aplikacije.

U oba slučaja servis objekt proširuje klasu Serviceu okviru AndroidSDK, i to kao samostalna komponenta ili dio aplikacije s kompletnim korisničkim sučeljem. Kreirani servis radi u procesu komponente koja ga je pozvala, dakle on ne započinje novi proces ili novi thread ukoliko drugačije nije navedeno.

U slučaju da servis radi veliki posao ili više njih istovremeno, onda se preporučuje stvaranje novih thread-ova unutar njega. Time se smanjuje rizik od pada aplikacije. Prije nego što se krene u detaljno objašnjenje kako kreirati servis, mora se prvo razumjeti na koji način servis ima interakciju s AndoridOS. Prilikom kreiranja servisa, navedeni servis mora se integrirati u manifest datoteci koju aplikacija koristi u okviru taga<service>. Također potrebno je definirati dozvole potrebne za pokretanje, zaustavljanje i povezivanje na servis, kao i neophodni poziv servisa. Nakon što je servis implementiran, servis se može koristiti upotrebom metode Context.startService(). Ako je usluga servisa već pokrenuta, kasnijom upotrebom metode startService(), usluga se ne pokreće ponovo. Servis funkcionira dok se ne pozove metoda Context.stopService() ili servis završi s radom i sam pozove f-ju za prestanak rada stopSelf(). Aplikacije koje žele da koriste servis moraju pozvati funkciju Context.bindService() za uspostavljanje veze sa servisom. Ako servis nije pokrenut, on se pokreće u tom trenutku, ako je servis već pokrenut, aplikacije mogu da šalju zahtjeve za izvršavanje određenih usluga ako zato imaju dozvolu.

Životni ciklus servisa



Životni ciklus servisa je jednostavniji od životnog ciklusa aktivnosti. Kod servisa treba više obratiti pažnju na njihovu kreiranje i uništenje jer korisnik najčešće nije svjestan njihovog funkcioniranja i uglavnom nema kontrolu nad njima. Za kreiranje servisa, neophodno je stvoriti klasu sa Service proširenjem i deklarirati ju u Manifest datoteci. U klasi treba implementirati **Callback** metode i definirati servis:

1. onStartCommand() – sustav poziva metodu kada neka od komponenti aplikacije zatraži stvaranje servisa pomoću metode startService(). Tada se kreira pokrenuti servis koji radi u pozadini nedefinirano dugo. Zato treba narediti servisu njegovo uništenje kada svoj posao odradi pomoću metodestopSelf() ili stopService() koja se poziva iz komponente. Kod vezanih(bind)servisa implementacija ovih metoda nije potrebna.
2. onBind() – sustav poziva ovu metodu kada neka komponenta zahtjeva kreaciju vezanog servisa pomoću metode bindService().Implementacija ove metode treba pružiti klijentima(komponentama koje ju pozivaju) sučelje preko koga će oni komunicirati sa servisom i to radi tako što metoda vraća vrijednost **IBinder**. Ova metoda mora uvijek biti implementirana, a ukoliko nije potrebno stvaranje vezanog servisa ona treba vratiti vrijednost **null**.
3. onCreate() – ova metoda se poziva odmah nakon što komponenta zatraži stvaranje servisa(prije nego što se onStartCommand() ili onBind() pozovu. U kodu ove metode se implementira sama svrha servisa. Ukoliko je servis već pokrenut , ova metoda se ne poziva.
4. onDestroy() – poziva se nakon što servis izvrši svoj zadatak (kroz stopService()), ili se odveže od svih komponenata (unbindService()) i uništava ga.

Implementacijom **Callback** metode mogu se uočiti dva ugnježđena životna ciklusa servisa:

1. cjelokupni život servisa nalazi se između poziva onCreate() i onDestroy() metoda. U onCreate() treba definirati servis i njegovo ponašanje a u metodi onDestroy() osloboditi sve resurse koje je on zauzeo. Bezobzira na to je li servis pokrenut ili vezan on mora biti kreiran i uništen.
2. aktivan život servisa počinje pozivom onStartCommand() ili onBind() metode. Ovim metodama se, prilikom pozivanja servisa, prosljeđuje **Intent** objekt kroz startService() ili bindService(). Aktivan život pokrenutog servisa se završava u istom trenutku kada i cjelokupan život, dok se kod vezanih servisa on završava pozivom onUnbind() metode.

Pokrenuti i vezani tip servisa nisu razdvojeni, tj. Servis pokrenut sa startService() može biti naknadno vezan sa komponentom

**Primjer**: možemo pokrenuti servis koji svira glazbu u pozadini koji se poslije može vezati za aktivnost Music player-a ukoliko korisnik to želi.

Servisi mogu obavještavati korisnika o svom radu na dva načina:

1. preko **toast** poruka.
2. preko obavijesti u statusnoj liniji i statusnom prozoru.

Toast poruke su kratke poruke u crnim pravokutnicima kojise pojavljuju na ekranu preko glavne aktivnosti i nestaju poslije nekoliko sekundi. Statusna linija se kod standardnog Android sučelja (i aplikacija koje nisu full-screen) nalazi na samom vrhu ekrana i moguće je njeno proširenje na statusni prozor. U statusnoj liniji se često mogu vidjeti obavijesti o napretku servisa(otpočeto preuzimanje sadržaja s interneta, preuzimanje je u toku, preuzimanje je završeno itd.)

U statusnom prozoru se može vidjeti više informacija o progresu servisa i eventualno reagirati s njihovim rezultatima (pokretanje ili pregled preuzetog sadržaja, otvaranje primljenog email-a u većem prozoru itd.)

# Prijemnici poruka

Prijemnici poruka (broadcast receivers) upravljaju komunikacijom između Android OS-a i aplikacija. Oni odgovaraju na poruke dobivene od drugih aplikacija ili sustava.

**Primjer**: aplikacije mogu uputiti poruku drugim aplikacijama obavještavajući ih da su neki podaci preuzeti, da se nalaze na uređaju i da su raspoloživi za korištenje.

Prijemnici poruka će prepoznati ovu komunikaciju i pokrenuti odgovarajuću akciju. Prijemnici poruka se implementiraju kao podklasa Broadcast Receiver klase, a svaka poruka je prijemnik poruke koji je Intent objekt.

**Primjer:**

**public class** MyReceiver extends Broadcast Receiver{**public void** onReceive(context, intent) {}}

Prijemnici poruka su komponente koje reagiraju na slanje obavijesti i najave u sustavu. Mnoge poruke stižu od samog sustava, kao što je obavijest da je baterija prazna, da je zaslon isključen, da je print screen naredba uspješno sačuvala sliku itd. Aplikacije isto mogu slati obavijesti, na primer kada uspješno skinu sadržaj sa Interneta i žele da obavijeste druge aplikacije da ga mogu koristiti. Iako prijemnici poruka ne mogu pružiti korisničko sučelje, oni mogu stvoriti obavijesti u statusnoj liniji i tako obavijestiti korisnika.

Prijemnici poruka najčešće samo signaliziraju drugim komponentama da počnu s izvršavanjem određenih zadataka i sami po sebi ne rade velike poslove. Prijemnici se implementiraju u Java klasama s proširenjem Broadcast Reciever i aktivira ih proslijeđeni Intent objekt, slično kao i kod aktivnosti.

Android OS karakterizira upotrebu međuprocesne komunikacije (IPC) u okviru same aplikacije kao i između aplikacija međusobno. Android OS svaku aplikaciju pokreće u zasebnom procesu (sandbox) s jedinstvenim identifikatorom(ID).

Svaka aplikacija ima svoju zasebnu instancu Dalvik VM. Na ovaj način aplikacije su u potpunosti nezavisne jedne od drugih s vlastitim adresnim prostorom. Ovakva implementacija omogućava sigurnost jer jedan proces ne može direktno upravljati memorijom drugog procesa. U mnogim slučajevima komunikacija između procesa je neophodna Zato OS mora osigurati mehanizme za međuprocesnu komunikaciju. Android OS sadrži jedan oblik međuprocesne komunikacije koji je karakterističan samo za njega, a to je Binder okvir. IPC na Android OS se većinom odvija preko Binder okvira i u maloj mjeri na soketima koji se koriste na najnižoj razini.

Međuprocesna komunikacija u Binder okviru je implementirana kao klijent-servis odnos.

Na Android platformi postoje tri mehanizma, tj. Implementacije međuprocesne nekomunikacije koja e odvija preko Binder okvira:

1. Intent,
2. Messenger,
3. Jezik za definiranje Android sučelja -AIDL

Messenger i AIDL implementacije rade s vezanim (bound) servisima, dok Intent objekti rade s pokrenutim(started) servisima. Tri od četiri tipa komponenti-aktivnosti, servisi i prijemnici poruka pokreću se pomoću asihrone poruke zvane Intent. Intent objekt spaja individualne komponente za vrijeme runtime-a bez obzira na to je li komponente pripadaju istoj aplikaciji. Intent objekt se može zamisliti kao glasnik koji od komponente zahtjeva određenu aktivnost. Intent se stvara kao apstraktni objekt u kome se definira poruka koja se šalje komponenti.

Pod apstraknim se misli na to da komponente, koje trebaju izvršiti akciju, ne moraju biti definirane u Intent objektu. Intenti se sastoje iz dva djela akcije i podataka.

Intent objekti se mogu zamisliti kao poruke koje se šalju širom OS kako bi izvršile određenu akciju na nekoj komponenti koja ima odgovarajući filter za tu akciju. Osnovne komponente svake aplikacije imaju definirane filtere u AndroidManifest.xml datoteci te aplikacije. Filter jedne komponente sadrži akcije koje podržava ta komponenta. Kada se pošalje intent, Android OS pronalazi sve komponente koje sadrže filter za tu akciju. Ukoliko postoji više komponenti sa tim filterom, Android OS prepušta korisniku da izabere željenu komponentu. Postoje dvije vrste Intent objekta eksplicitni i implicitni.

Eksplicitni gađaju određenu komponentu po imenu i ograničenje samo na komponente unutar jedne aplikacije.

Implicitni, pokušavaju da pronađu komponente po akciji i nisu ograničeni na komponente jedne aplikacije. Intent objekt proslijeđen aktivnostima i servisima definira akciju koju treba izvršiti (prikazati ili poslati nešto), može sadržavati URI podataka nad kojima ona treba biti izvršena.

**Primjer:**

Intentom se od aktivnosti može zatražiti da prikaže određenu sliku ili otvori određenu web stranicu.

U nekim slučajevima je moguće pokrenuti aktivnost kako bi ona vratila rezultat, kroz Intent objekt (pitati korsinika da odabere neku osobu iz kontakta i vraćeni Intent objekt će sadržati URI tog kontakta). Intent objekt proslijeđen prijemniku poruka samo definira poruku koja će biti prikazana kao obavijest.

1. Dobavljači sadržaja

Dobavljači sadržaja se ne aktiviraju Intent objektima, već zahtjevima koji Content Resolver može uputiti.

Content Resolver kontrolira sve direktne transakcije između komponente i dobavljača kako komponenta ne bi morala to raditi. Ona samo poziva metode iz ContentResolver objekta i tako se stvara apstraktni odnos komponenta-dobavljač radi sigurnosti. Objekt Intent je pasivna struktura podataka koja drži apstraktni opis operacija koje se trebaju izvršiti, ili u slučaju broadcast receiver , opis nečega što se dogodilo ili se objavljuje.

Svaki tip komponenti se aktivira posebnim metodama:

1. Aktivnost se pokreće (ili joj se zadaje novi zadatak) prosljeđivanjem Intent objekta metodi startActivity(). Kada se od aktivnosti očekuje da vrati rezultat, on se prosljeđuje metodi startActivityForResult().
2. Servis se pokreće (ili mu se zadaju nove instrukcije ukoliko je već pokrenut) prosljeđivanjem Intent objekta metodi startService(). Servis se može vezati prosljeđivanjem Intent objekta metodi bindService().
3. Emitiranje se može inicirati prosljeđivanjem Intent objekta metodi sendBroadcast(), sendOrderedBroadcast() ili sendStickyBroadcast().
4. Dobavljaču sadržaja se može postaviti upit pozivanjem metode query() kod ContentResolver-a.

# Provideri sadržaja

Provideri sadržaja (content providers) upravljaju dijeljenim podacima aplikacije.

Ova komponenta (na zahtjev) osigurava podatke iz jedne aplikacije drugima.

Podaci se mogu nalaziti u bazi podataka, datotečnom sistemu ili na nekom drugom mjestu.

Provider sadržaja se implementira kao podklasa ContentProvider klase, i mora implementirati standardni skup API-a kako bi omogućio drugim aplikacijama izvršenje transakcija.

**Primjer**:

**public class** MyContentProviderextends ContentProvider{**public void** onCreate() {}}

Postoje dodatne komponente koje se mogu koristiti u razvoju aplikacija, odnosno pripadajućih entiteta, njihove logike i međusobnog povezivanja.

1. Dodatne komponente

Uz Aktivnosti, servise, prijemnika poruka, dobavljača sadržaja i providera sadržaja još preostale komponente su:

1. Fragmenti(fragments) – predstavljaju dio korisničkog sučelja u nekoj aktivnosti
2. Pogledi(views) – elementi korisničkog sučelja koji su iscrtani, odnosno predstavljeni na zaslonu, kao što su gumbi, liste, kvadratići za izbor (checkbox)
3. Prikazi(layouts) – hijerarhija pogleda koja kontrolira format zaslona i izgled pogleda
4. Namjere(intents) – poruke koje povezuju komponente
5. Resursi(resources) – eksterni elementi
6. Manifest–konfiguracijska datoteka za aplikaciju

Struktura Android projekta se sastoji od nekoliko mapa i datoteka:

1. build – sadrži generirane datoteke Build konfiguracija i R (R.java)
2. lib – mapa za biblioteke potrebne za razvoj android aplikacija
3. src – sadrži izvorne (.java) datoteke projekta. Ovdje se nalazi datoteka MainActivity.java koja sadrži klasu aktivnosti, a koja se izvršava pri pokretanju aplikacije.
4. res – sadrži slike, prikaze, vrijednosti i Android manifest datoteku
5. res/drawable – mapa za slike, odnosno objekte dizajnirane za zaslone
6. res/layout – mapa koja sadrži datoteke koje određuju korisničko sučelje aplikacije
7. res/menu – mapa za objekte menija koji su dizajnirani za izgradnju menija u aplikaciji
8. res/values–mapa za razne XML datoteke koje sadrže resurse, kao što su stringovi i boje
9. AndroidManifest.xml–manifest datoteka koja opisuje bitne osobine aplikacije i definira svaku njenu komponentu