רכיב הסרוויס

הקדמה

הפרוסס (תהליך) הוא המסגרת החיצונית, הוא התהליך של מערכת האנדרואיד, הוא כמו אינקובטור והוא זה שעובד מול המערכת. ברגע שנכנסת לתוכו תוכנית, יש לה מעטפת ריצה והיא נותנת משמעות לפרוסס.
לכבות אותו זה לכבות את המכשיר, ממנו נולדים כל הפרוססים.
המעטפת, הפרוסס תמיד קיים- אבל לפעמים הוא ריק והוא מחזיק את המסגרת שבה עתידים להתקיים אפליקציות וכדומה. בפרוסס יכול להיות חי או חלול, אז הוא קיים כמו זובי, ובמקרה כזה הפרוסס ימוחזר.
הפרוסס- הוא האינקובטור, אם יש מיין טרייד פועל ואני רוצה להפעיל עוד תכנית קטנה לא צריך עוד פרוסס אלא הוא יפעל המקבילית למיין טרייד ויחזור למיין טרייד. אם המיין טרייד נסגר כל התוכניות הקטנות והמקבילות יסגרו, לא יהיה להם לאן לחזור.
המיין פרוסס הוא הראשי ומחזיק בתוכו בין כל הטריידים- את אחד העיקריים שהוא היואייטרייד ואחד התפקידים שלו הוא להחזיק את מסך המגע, הוא עובר בלופים וכל הזמן שואל אם יש לו משהו לאסוף מהמסך או לשפוך על המסך.
הטריידים מתקיימים גם כמה במקביל ומתקשרים ביניהם על פי דרישת התוכנית.

הקרנל הוא המתווך בין המידע שבמערכת מבקשת לשדר ובין המכשיר שעתיד לשדר אותו, הקרנל עטוף בדרייברים שכל אחד מהם הוא מתאם אחר.

ה **UI Thread** מציג על המסך ואוסף אירועי נגיעות על המסך.
הטרייד שלי באקטיביטי הופך להיות משיק, וכל פעם שיש משהו על המסך הוא מספר ליואייטרייד והוא שופך את זה למסך, כל העניין הוא התקשורת בין הטריידים.
היוואייטרייד והמיין טרייד הם מתקשרים אחד עם השני את מה שהאקטיביטי רוצה לשדר.
האפליקציה הנוכחית קיבלה רשות ליצור קשר עם היואייטרייד להציג על המסך.
השיטות ליצירת קשר עם ה main Thread :

1. תורשה מטרייד.
2. טרייד יכול לשמש מעטפת ל runAble, interFace לכך שמשהו הוא בר הרצה.
   אימפלמנט לאינטרפייס ראנאייבל. ובסוף ניתן לו ראן.
3. פוסט – באמצעות הנדלר.
   הנדלר במקום שאני פותח אותו הוא הופך להיות קשור אליו.
   קובעים איזה מקום להנדלר וזה למעשה יצירת צינור לקשר שבין המיין ליואיי טרייד. וכל מי שרוצה יכול לשלוח לי דברים דרך הצינור. כעקרון הוא חד כיווני.
   ההנדלר יוצר עוגן עם המיין אקטיביטי ותופס מקום ריצה.
   <u>יש להנדלר 2 תכונות:</u>
   אפשר לשלוח אליו הודעות ואפשר לשלוח לו פוסט של ראנאייבל.
   בגלל שהנדלר הוא עוגן שנמצא במיין צריך להעביר לו רק סניפט והוא כבר יודע איפה לשים את זה.
   לכל view יש הנדלר משלו שנוצר יחד איתו כשיוצרים איתו את המיין טרייד.
   אפשר לבקש פוסט והפוסט של הוויו ילך למיין טרייד.
   שינויים באלמנטים ויזואליים נשתמש בפוסט.
   פוסט דילאיי- פוסט עם דילאיי מסוים.
4. לשלוח הודעות להנדלר (השיטה השנייה של הנדלר) שימוש בניו מסג' שזה קלאסס של ג'אווה. אם מכניסים את הריפלאיי טו הוא יוצר את ההנדלר כצינור דו כיווני ומאפשר דיאלוג הודעות.
   היתרון של השיטה זאת הוא שאפשר להעביר מגוון דברים ולא להעביר קוד ראנאייבל וזה מאפשר לתקשר בין שני אקטיביטיס שנמצאים בפרוססים שונים.
   שיטת המסג' היא בכלל טובה ויעילה להעברת מידע בין טריידים שונים.
   ברגע שהוא מקבל הודעה הוא שואל וואט?? (מה?) מבין ביט מפ וממשיך בתהליך.
5. באייסינקטאסק, ברקע נעשית ההורדה של התמונה ובהוצאה לפועל- שאוטומטית ביואייטרייד מזינים את התמונה לוויואימג'.

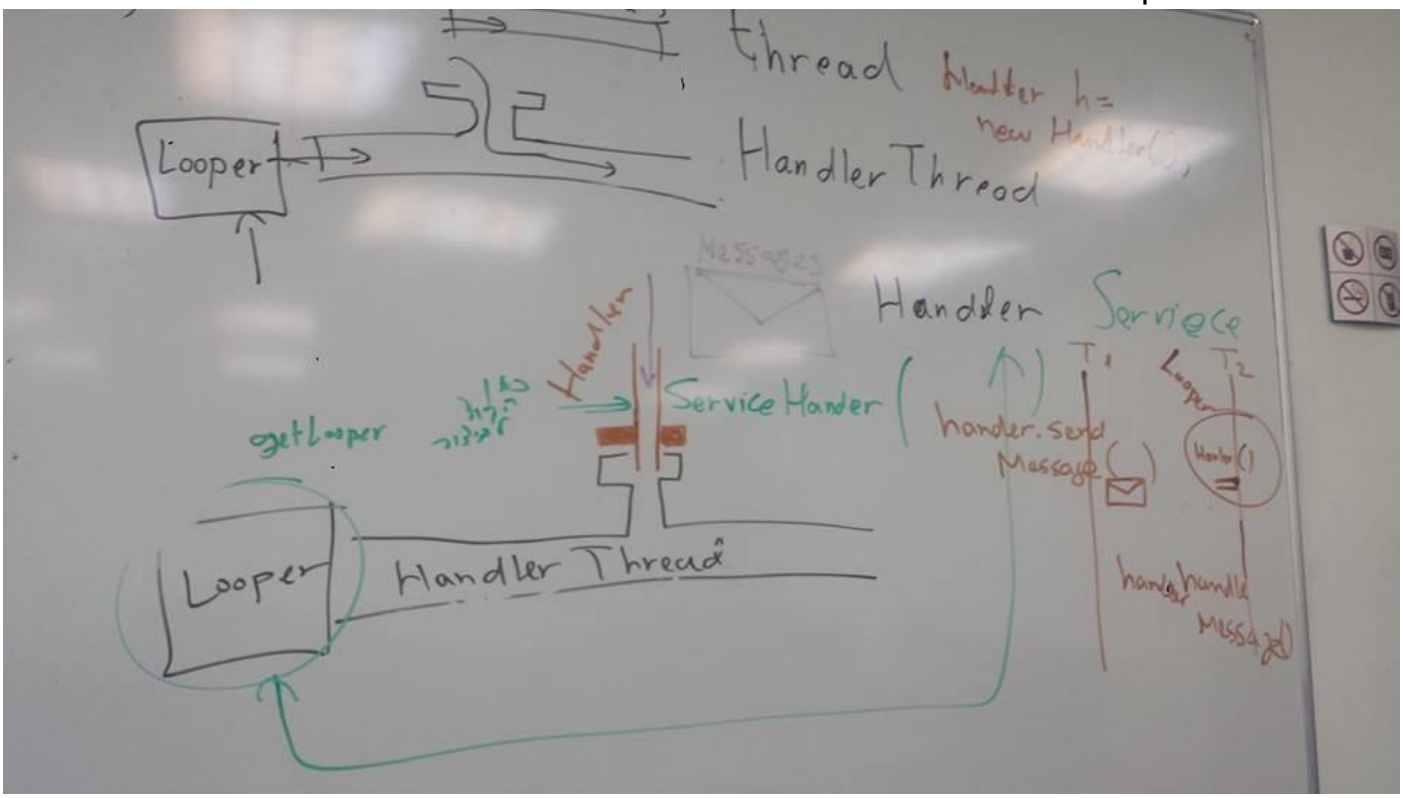
Service

לסרוויס, כמו לאקטיביטי יש מחזור חיים וזה מאפשר לשמור מידע בהפסקות.
כשיש כמה פניות במקביל לרשת.
סרוויס הם רכיב אפליקציוני ואם רוצים להגדיר סרוויס יש להגדיר אותו במניפסט, כמו שמגדירים אקטיביטי. יש כמה סוגים של סרוויס.
מאפשר עיבוד מידע ארוך וטווח והרבה פעמים יש חשיבות לעובדה שאפשר לעצור באמצע ואחר כך לחזור מהנקודה שהפסקנו.
דבר נוסף שיש לסרוויס ואין לטרייד זה האפשרות לבצע תקשורת בין תהליכים שונים באמצעות סרוויס.
למשל תקשורת בין מספר אפליקציות.

<u>משפחות הסרוויס:</u>
קיימים 2 סוגי של משפחות-

1. אינטנט סרויס- מאותחלים עם סטרט סרוויס אפפ סרוויס או רינטנט סרוויס.
   תחתיו יש 2 אימפלמינטציות-
   ההבדל ביניהם שאינטנט סרוויס הוא מסתנכרן והסרוויס הרגיל אינו מסתנכרן וכולם באותו זמן יכולים להיכנס ולדרוס באותו הזמן. כלומר אם הוא קיבל את אותה ההודעה/ פקודה פעמיים הוא בנוי לכך שאם הוא בפעולה הוא ידחה בקשות אחרות זהות—זה הלא מסונכרן.
   - להכניס את השקף – מספר 8.

באון ביינד להחזיר נאל!!!



הסרוויס הוא רכיב אפליקציוני שנועד לבצע פעולות ארוכות ברקע מבלי להחזיק את המסך של המשתמש, כלומר, בזמן שהסרוויס רץ המשתמש יכול לעשות דברים אחרים באפליקציה.
הסרוויס יכול להיקשר לשירות קיים, לתקשר בין תהליכים או להתמודד עם עסקאות ברשת.
אזהרה: ברירת המחדל של הסרוויס היא שירוץ ב Main thread על הפרוסס הראשי. כלומר, הוא מחזיק את המסך של המשתמש עד שהוא מסיים את הפעולה שלו. לכן, אם הסרוויס נועד לעשות עבודה אינטנסיבית או לחסום פעולות, צריך ליצור טרייד חדש בו תתבצע עבודת הסרוויס. וכך, ע"י השימוש בטרייד נפרד מופחת הסיכון שאפליקציה לא תגיב ושיתרחשו טעויות. וכך ה- Main thread של האפליקציה נשאר פנוי עבור האינטרקציה של המשתמש עם האקטביטי.

<u>סרוויס הוא לא:</u>
סרוויס הוא **לא** תהליך (process) נפרד ואינו מריץ תהליך משל עצמו, (פרט למקרים מסוימים) הסרוויס רץ באותו תהליך שהאפליקציה רצה בו.
סרוויס הוא **לא** thread.
<u>מהו סרוויס – שני תכונות עיקריות:</u>

1. הסרוויס משמש עבור האפליקציה כמתקן שעובד מול המערכת ומודיע לה על דברים שצריכים להיעשות ברקע באמצעות:
   Context.stertService()
   תפקידו לבקש מהמערכת שתסדר לסרוויס את לוח הזמנים או סדר פעולות הריצה עד שהסרוויס או מישהו אחר יעצרו אותו מפורשות.
2. הסרוויס משמש עבור האפליקציה כמכשיר שבאמצעותו היא יכולה לתקשר עם אפליקציות אחרות באמצעות:
   Context.bindService()
   מאפשר ליצור זמן ריצה ארוך מבלי ליצור אינטרקציה עם ה- UI.

כשרכיב הסרוויס נוצר בפועל, מאחת הסיבות הללו, הדבר היחיד שהמערכת עושה זה להתייחס לרכיב ולקרוא למתודה **onCreate()** ולכל הקולבקים שב main thread.
הדברים הללו עולים לסרוויס שיממש אותם כראוי.

באופן כללי ניתן להפעיל את הסרוויס מתוך אפליקציות אחרות או לחסום אפשרות זו באמצעות קובץ המניפסט.

קיימות שתי צורות לסרוויס:

הכנה	extends service / IntentService	
	Start service	**Bound service**
לידה	רכיב אפליקציוני קורא להתחיל את הסרוויס באמצעות אחת המתודות: startService() או onStartCommand()	bindSercive() או onBind()
מהלך חיין	הסרוויס רץ כל עוד הוא לא סיים את ה**משימה** שלו. אפילו אם הרכיב שהפעיל אותו ימות, הסרוויס ימשיך לרוץ.	**נשלט** ומסוגל לקיים אינטרקציה עם השירות, לשלוח בקשות, לקבל תוצאות וכו'
הרוצח	לרוב **הסרוויס** יהרוג את עצמו עם סיום המשימה באמצעות stopSelf()	כשכל הרכיבים הכבולים אליו מתנתקים ממנו על ידי onBind() הסרוויס מת **מהבדידות**. unBind()
	או ש**רכיב** (זה שהפעיל את הסרוויס או רכיב אחר) יפסיק את הסרוויס באמצעות stopService().	
	או במצב שבו כמות המשאבים של **מערכת האנדרואיד** נמוכה ואז המערכת תשמיד את הסרוויס ותחיה אותו אוטומטית כאשר זמינות המשאבים שוב תגדל = נשלוט בתחיה שלו באמצעות הערך החוזר מהפונקציה onStartCommand(). וגם, אם הסרוויס רץ על ממשק המשתמש (ולא ברקע) הוא יקבל חשיבות גדולה יותר בעיני מערכת האנדרואיד ופחות סביר שהיא תשמיד אותו במצב של מצוקת משאבים.	

המתודות העיקריות:

onStartCommand()

נקרא למתודה זו כאשר נרצה שרכיב אחר יתחיל את הסרוויס.
מרגע זה, הסרוויס מתחיל ורץ ברקע ללא הגבלת זמן. באחריות המתכנת להפסיק את הסרוויס כהוא מסיים את עבודתו ע"י קריאה למתודה stopSelf() או stopService().
רלוונטית רק לסרוויס מסוג Start service.

onBind()

נשתמש במתודה זו כאשר נרצה שרכיב אחר יפעיל את הסרוויס וישלוט בו ע"י שימוש ב- bindSercive(). חובה לספק למשתמש ממשק דרכו הוא יוכל ליצור תקשורת עם הסרוויס ע"י קביעת return לפונקציה – iBinder. במקרה שלא מרצה לאפשר bound service נקבע את ערך ה- return ל- null.

onCreate()

במסגרת מתודה זו ניצור את הסרוויס ונבצע התקנה חד פעמית. רלוונטית רק למצב שבו הסרוויס אינו קיים.
זהו שלב שקודם ל- onStartCommend() או onBind().

onDestroy()

מתודה הנקראת כאשר הסרוויס מסיים את המשימה שלו. יחד עם השמדת הסרוויס, המתודה גם מנקה כל משאב שקשור לסרוויס כמו טריידים, ליסטנרים, רסיברים וכו'

Manifest

נוסיף לקובץ המניפסט תגית service.
בתוך התגית נגדיר מספר תכונות, החשובות שבהם:

```
< service android:name="ExsempleService1"
```

זאת התכונה ההכרחית היחידה- היא מפרטת את שם המחלקה של הסרוויס.
מרגע שהאפליקציה פורסמה, אסור לשנות שם זה, כי יש תלות באינטנט מפורש ליצירת start service או bound service ושינוי השם מביא סיכון גדול שהקוד יישבר.
ליצירת אפליקציה מאובטחת, תמיד נשתמש באינטנט מפורש לסרוויס (start & bound) **ולא** נצהיר על אינטנט פילטר לסרוויס. אבל זה ממש קריטי שתאפשר עמימות מסוימת כמו להחליף סרוויסים להתחלה, אז נשתמש אינטנט פילטר לסרוויס ונכלול בו את שם הרכיב מהאינטנט, נגדיר את הפקג' לאינטנט עם setPackage() אשר מספק הגדרה למטרת הסרוויס.

```
android:exported="false" />
```

באמצעות תכונה זו נוכל להגדיר אם הסרוויס ניתן להפעלה על ידי אפליקציות אחרות או מוגבל לאפליקציה שלנו בלבד. הערך false ימנע מאפליקציות אחרות להפעיל את הסרוויס אפילו כשמשתמשים באינטנט מפורש.

Start service- תהליכים ורכיבים.

סרוויס זה נוצר על ידי רכיב כלשהוא. כשהסרוויס מתחיל, יש לו מחזור חיים בלתי תלוי ברכיב שהתחיל אותו והסרוויס יכול לרוץ ברקע ללא הגבלת זמן. אפילו אם הרכיב שהתחיל אותו הושמד. ניתן לעצור את הסרוויס בשני דרכים: הסרוויס עוצר את עצמו כשעבודתו מסתיימת (stopSelf()) או שרכיב אחר יכול לעצור אותו (stopService())

חשוב להבין: הסרוויס רץ על אותו התהליך המוצהר של האפליקציה ובאותו Main thraed של האפליקציה כברירת מחדל של הסרוויס. לכן אם ביצועי הסרוויס אינטנסיביים או חוסמים פעולות בזמן שהמשתמש באינטרקציה עם אקטיביטי מאותה אפליקציה, הסרוויס יאט את ביצועי האקטיביטי. כדי להימנע מהאטת הביצועים צריך להתחיל טרייד חדש בתוך הסרוויס וכך הסרוויס יפעל ברקע ולא יפריע לחוויית המשתמש.

התחלת הסרוויס - כשאנו קוראים למתודה startService אז מערכת האנדרואיד קוראת למתודת הסרוויס onStartCommend() ומעבירה את האינטנט (לעולם לא נקרא לonStartCommend() ישירות). במידה וזו הפעם הראשונה, המערכת תזמן קודם את onCreate() עם אתחול הסרוויס באמצעות רכיב, נעביר גם אינטנט שמפרט את הסרוויס וכל הנתונים הנדרשים לו.
במידה ונרצה שהסרוויס יחזיר אילו תוצאות, וכך הרכיב שהפעיל את הסרוויס יוכל ליצור אינטנט תלוי לשידור (עם getBroadcast()) ומשם לשגר לסרוויס יחד עם האינטנט שהתחיל את הסרוויס. במקרה זה הסרוויס ישתמש בשידור כדי לשלוח תוצאות.
הפסקת הסרוויס -הסרוויס חייב לנהל את מחזור החיים של עצמו. המערכת לא עוצרת או משמידה את הסרוויס אלא אם כן היא חייבת לאוושש את משאבי זיכרון המערכת והסרוויס ממשיך לרוץ אחרי onStartCommend חוזר. אז, הסרוויס חייב לעצור את עצמו ע"י קריאה ל stopSelf() או שרכיב אחר יכול לעצור אותו ע"י קריאה ל stopService(). בקריאה הראשונה לאחת המתודות הללו המערכת תשמיש את הסרוויס בהקדם האפשרי.
אולם, אם הסרוויס מתמודד עם בקשות מרובות לonStartCommend במקביל, אז נדרש מאיתנו להגדיר לסרוויס לעצור עם סיום המשימה וזאת, מאחר וייתכן שמאז התקבלה בקשה חדשה לסרוויס (עצירה בסוף של הבקשה הראשונה יגרום לסיום הבקשה השנייה). כדי להימנע מבעיה זו, ניתן להשתמש לstopSelf עם פרמטר אינטג'ר כדי לוודא שהבקשה לעצור את הסרוויס תמיד תהיה מבוססת על בקשת ההתחלה המוקדמת ביותר. נעביר את ה ID של בקשת הסרוויס משם הוא נשלח ל onStartCommend לבקשת הסטופ המקבילה. ואז אם הסרוויס קיבל בקשת סטרט חדשה לפני שהיה אפשר לקרוא לסטופסלף אינט'- אז האיידי לא יתאים והסרוויס לא יעצור.
אזהרה: זה חשוב שהאפליקציה עוצרת את הסרוויס כשהוא מסיים לעבוד, כדי להימנע מבזבוז משאבי מערכת ולשמור על כוח הסוללה. אם הכרחי, רכיבים אחרים יכולים לעצור את הסרוויס ע"י קריאה לstopService. אפילו אם מתאפשר לך ליצור bound לסרוויס, אתה חייב תמיד לעצור את הסרוויס בעצמך למקרה שאי פעם תתקבל קריאה לonStartCommend .

על מנת להשתמש ב service נוכל לרשת אחת מהמחלקות הבאות:

-extend Service

זוהי המחלקה הבסיסית לכל הסרוויסים. כשיורשים ממחלקה זו, חשוב ליצור טרייד חדש שיבצע את כל עבודת הסרוויסים, אז הסרוויס ירוץ ברקע ורמת הביצועים לא תיפגם.
מאחר ואנו מתמודדים עם כל קריאה ל ()onStartCommend בעצמנו, אז ניתן לטפל במקביל במספר בקשות לסרוויס: נייצר טרייד חדש עבור כל בקשה ונריץ אותם באופן מידי (לא נחכה בתור שהבקשה הקודמת תסתיים).
נשים לב ש()onStartCommend חייב להכיל אינטג'ר. האינטג'ר הוא ערך המתאר איך המערכת אמורה להמשיך את הסרוויס במקרה שהמערכת הורגת אותוץ.
הערך החוזר ()onStartCommend חייב לביות אחד מהבאים:

START_NOT_STICKY - אם המערכת הורגת את הסרוויס אחרי החזרה מ ()onStartCommend לא ניצור מחדש את הסרוויס, אלא אם כן יש אינטנטים עומדים למשלוח. זוהי האופציה הבטוחה ביותר כדי להימנע מלהריץ את הסרוויס כשלא נחוץ וכשהאפליקציה יכולה בפשטות לאתחל מחדש כל עבודה לא גמורה.

START_STICKY- אם המערכת הורגת את הסרוויס אחרי ()onStartCommend חוזר, אז תהיה יצירה מחדש של הסרוויס וקריאה ל()onStartCommend . במקרה זה לא נשלח את האינטנט האחרון, אלא, המערכת תקרא ל()onStartCommend עם אינטנט= null, אלא אם כן היו אינטנטים תלויים ועומדים להתחלת הסרוויס, בכל מקרה, האינטנטים הללו יצאו למשלוח. זה מתאים למדיה פלייר שאינם מבצעים את הפקודה אך רצים ללא הגבלת זמן ומחכים לעבודה.

START_REDELIVER_INTENT- אם המערכת הורגת את הסרוויס אחרי ()onStartCommend חוזר, אז ניצור מחדש את הסרוויס ותקרא ל()onStartCommend עם האינטנט האחרון שנשלח לסרוויס. כל אינטנט תלוי ועומד נשלח בתור. זה מתאים לסרוויס עם ביצועים פעילים עבור משימה שאמורה להיות מחודשת מידית כמו הורדת קובץ.

-extend IntentService

זוהי תת מחלקה של סרוויס ומשמשת כ worker thread ומאפשרת להתמודד עם מספר בקשות סרוויס, עם זאת, חשוב לדעת שמדובר בטיפול בכל בקשה בנפרד לפי הסדר ו**לא** ניתן לטפל בכמה בקשות בו זמנית.

כעת נממש את onHandleIntent() והוא מקבל אינטנט לכל בקשה לסרוויס כך שהסרוויס מתבצע ברקע.

האינטנט סרוויס עושה את הדברים הבאים:

1. יצירת טרייד נפרד מהטרייד הראשי של האפליקציה ובו יוצאו לפועל כל משלוחי האינטנט ל- onStartCommend().
2. יוצר תור מסודר שמעביר אינטנט אחד בכל פעם ל- onHandleIntent() וכך מסיר מעלינו את הסכנה שבבקשות מרובות (=מספר סרוויסים במקביל).
3. ברגע שבקשת סרוויס אחת הושלמה האינטנט סרוויס מפסיק את הסרוויס שסיים אוטומטית, כך שלעולם לא צריך לקרוא ל- stopSelf().
4. אימפלמנט שקובע עבור המתודה onBind() ערך חוזר null.
5. אימפלמנט של onStartCommend() ששולח את האינטנט אחד אחד לתור ודואג להעבירם לפי התור ל- onHandleIntent() שייושמו.

onHandleIntent()- על המתכנת ליישם את onHandleIntent() כך שיוכל לקלוט את ההוראות מהרכיב שהפעיל את הסרוויס וגם צריך לספק קונסטרקטור לסרוויס.
במידה ונרצה ליישם גם מתודות callback כמו onCreate, onStartCommend או onDestroy אז ממש הכרחי ליישם את הסופר. אם לא, עלולות להיווצר תקלות בהפעל הטרייד החיצוני על ידי האינטנט סרוויס.

onBind()- במתודה זו אין צורך לקרוא לסופר אלא רק להחליט את הסרוויס ניתן לכבילה או לא (bound).

יצירת bound Service:

זהו סוג של סרוויס המאפשר להיקשר אליו ע"י קריאה ל ()BindService במטרה ליצור התקשרות ארוכת טווח.
נשתמש בסרוויס זה כאשר נרצה ליצור אינטרקציה בין הסרוויס לאקטיביטי או לרכיבים אפליקציונים אחרים באפליקציה שלנו. וגם במצבים בהם נרצה לאפשר לאפליקציות חיצוניות להשתמש ולהפעיל חלק מהפונקציונליות של הסרוויס.

חובה לעשות אימפלמנט למתודה ()onBinder ולהחזיר ממנה iBinder – הוא זה שמזהה את הממשק לתקשורת עם הסרוויס.
רכיבי אפליקציה אחרים יכולים אז לקרוא ל ()bindService כדי לשלוף את הממשק ולהתחיל לקרוא למתודות שבסרוויס.
הסרוויס חי רק כדי לשרת את הרכיבים שהוא כבול אליהם, כך שכאשר אין רכיבים הכבולים לסרוויס, המערכת משמידה אותו זה הבדל בין ()bindService ל ()onStartCommend- ששם חייב להגדיר לסרוויס לעצור עם סיום המשימה וכאן המערכת משמידה אותו ברגע שאין רכיבים הכבולים אליו.

על מנת ליצור bound service, הדבר הראשון שחייב לעשות הוא לזהות את האינטרפייס שמפרט איך הלקוח יכול לתקשר עם הסרוויס. אינטרפייס זה שנמצא בין הסרוויס ללקוח חייב להיות אימפלמנט של iBinder וזה מה שהסרוויס חייב להחזיר ממתודת הקולבק ()onBind. מרגע שהלקוח קיבל את ה iBinder ניתן לתקשר עם הסרוויס דרך האינטרפייס.

רכיבים רבים יכולים ליצור bound לסרוויס בבת אחת. כשרכיב כלשהוא יוצר אינטרקציה עם הסרוויס, זה קורא ל ()onBindService כדי להתנתק. ברגע שאין אף רכיב כבול לסרוויס, המערכת משמידה את הסרוויס.

הרצת סרוויס ברקע

דיברנו על מצב שבו אנחנו רוצים שהסרוויס ירוץ ברקע, כעת נעסוק במצב בו נרצה שהסרוויס ירוץ דווקא בחזית.
במצב זה עלינו לספק הודעות לסטטוס בר שמתחלף תחת onGoingHading שמשמעותו היא שההודעה לא יכולה להימחק אלא אם כן הסרוויס עצר או הוסר מהמסך.
()startForGround- נשתמש במתודה זו כאשר נרצה שהסרוויס ירוץ בממשק המשתמש. מתודה זו לוקחת שני פרמטרים, אינטג'ר שמזהה באופן ייחודי את ההודעה (=חייב להיות שונה מ -0) וההודעה לסטטוס בר.
()StopForGround - משמשת להסרת הסרוויס מהחזית. מתודה זו לוקחת ערך בוליאני, המצביע על אם להסיר את הסטטוס בר, גם מתודה זו אינה עוצרת את הסרוויס. בכל אופן, אם מפסיקים את הסרוויס באמצע ריצה במסך, אז ההודעה גם כן מוסרת.

ניהול מחזור החיים של הסרוויס

רק בשלבי החיים onResume ו onPaused הוא נמצא ב UI thread- בכל שאר הזמן הוא בmainThraed.
onResive- כשהוא מקבל את האינטנט, אם לא קורה עדיין כלום הוא חוזר למיין טרייד.

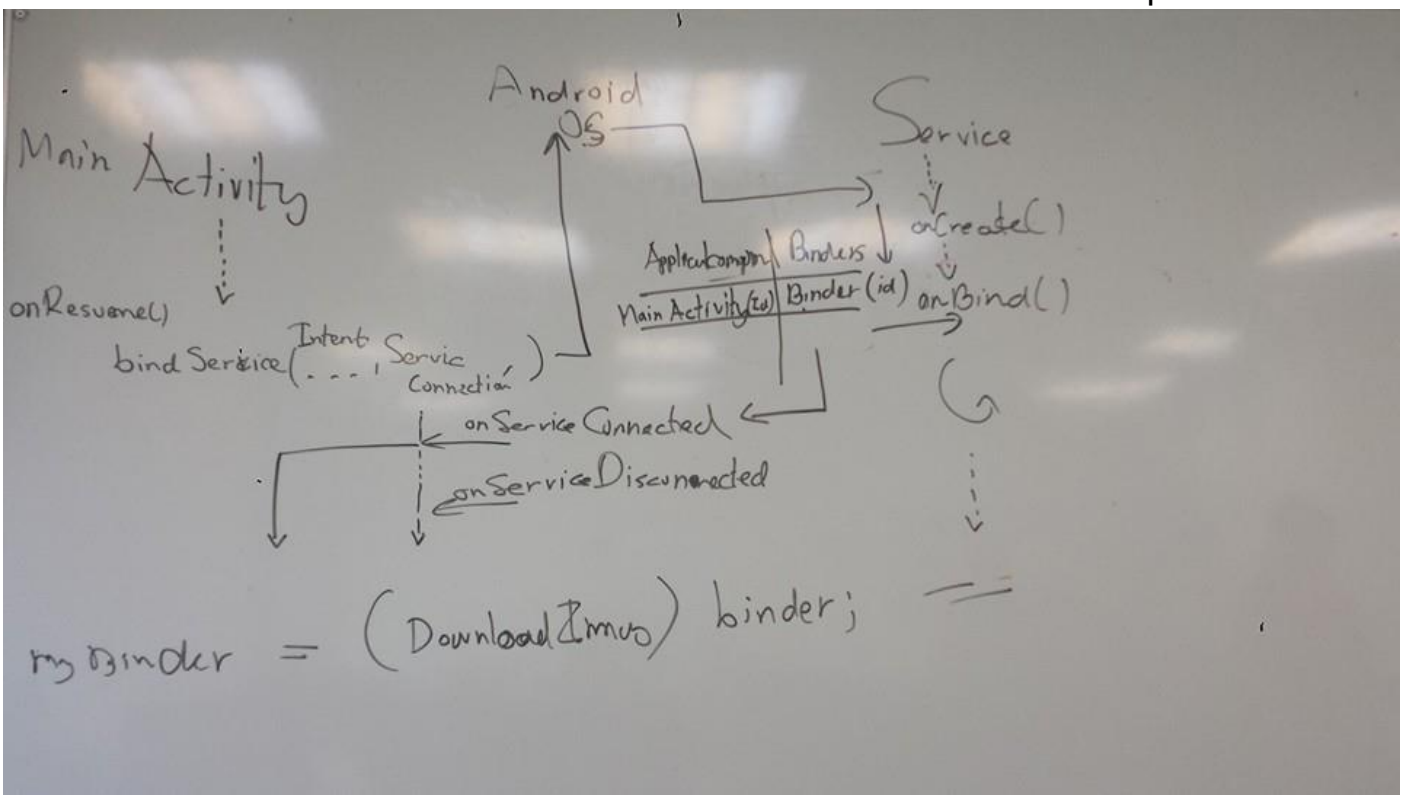
מחזור החיים-
ביינד סרוויס- חי רק אם יש מישהו שרוצה להתחבר איתו.

בסטרטסרוויס אנחנו לא יודעים מתי הוא מסיים כדי להמשיך את הפעולות, אך מתי נשתמש בבינד ומתי בסטרט?
אם אני יודע שאני מצפה לתשובה באקטיביטי שלי נשתמש בבינד ואם לא אז נשתמש בסטרט.

בדוגמא הורדנו תמונה מהרשת והשתמשמנו בהנדלר ואנחנו רוצים שבסופו של תהליך הוא יודיע שהתמונה ירדה.
און הנדל אינטנט – המסנג'ר הוא אובייקט שעוטף את ההנדלר ומאפשר את הדו שיח הדו כיווני.
בסוף ההורדה של התמונה – מסג'. סנט והוא מודיע שתהליך ההורדה הסתיים ושולח איתו את הקוד והביט מפ.
שולח את זה להנדלר שהוא עוגן.

האם באון הנדל אינטנט יכולתי לעשות בט'רייד?
ההנדלר מאחר וההנדלר נוצר במיין אקטיביטי בזמן שהיה במיין ט'רייד מחובר ליואייטרייד אז הוא תלוי בו והחזרה תהיה למיין אטיביטי.
סרוויס הוא אקטיביטי ללא יואיי- ללא מסך.

מחזור החיים של הסרוויס פשוט יותר מזה של האקטיביטי. בכל אופן, זה אפילו חשוב יותר שתשים לב מקרוב לדרך שבה הסרוויס שלך נוצר ומושמד, בגלל שסרוויס יכול לרוץ ברקע מבלי שהמשתמש יהיה מודע לכך.
מחזור החיים של הסרוויס- מרגע שהוא נוצר ועד שהוא מושמד- יכול לעבור בשני נתיבים שונים:
<u>Start service</u>- נוצר כאשר רכיב אחר קורא לאתחול הסרוויס. אז הסרוויס רץ ללא הגבלת זמן וחייב לעצור את עצמו ע"י קריאה ל ()stopSelf. או עצירה על ידי רכיב נוסף ע"י קריאה ל stopService. כאשר הסרוויס עוצר, המערכת משמידה אותו.
<u>Bound Service</u>- הסרוויס נוצר כאשר רכיב אחר קורא ל bindService . אז הרכיב מתקשר עם הסרוויס דרך אינטרפייס iBinder. הרכיב יכול גם לסגור את התקשורת ע"י קריאה ל onBindService. רכיבים מרובים יכולים להיכבל לאותו סרוויס וכאשר כולם מתנתקים המערכת תשמיד את הסרוויס- הסרוויס לא אמור להפסיק את עצמו.
שני הנתיבים הללו הם לא לגמרי נפרדים. ולכן, אתה יכול לכבול סרוויס שכבר התחיל עם startService ולאחר שהוא התחיל לרוץ אפשר לכבול אליו עם bindService. במקרה כזה stopService או stopSelf לא ממש יעצרו את הסרוויס אלא עד שכל הרכיבים יתנתקו.



אימפלמנטציה למתודות קולבק של מחזור החיים

כמו לאקטיביטי, גם לסרוויס יש מתודות קולבק של מחזור החיים שלו שניתן ליישם לצד שינויים במצב הסרוויס ולבצע עבודה בזמן הנכון.
ע"י יישום המתודות של מחזור החיים, ניתן להציג שני לולאות מקוננות על מחזור החיים של הסרוויס.
כל מחזור החיים של הסרוויס מתרחש בין הזמן שבו onCreate נקראת ועד onDestroyd מחזיר. כמו אקטיביטי, הסרוויס עושה את ההתקנה הראשונית בonCreate ומשחרר את כל התהליכים הנותרים במשאבים ב onDestroyd .
זמן החיים של הסרוויס מתחיל עם הקריאה ל onStartCommend או onBind. כל אחת מהמתודות בידיים של האינטנט שהועבר לstartService או bindService. בהתאמה אם הסרוויס התחיל, זמן החיים של האקטיביטי מסתיים באותו זמן שכל זמן החיים מסתיים (הסרוויס עדיין פעיל אפילו אחרי ש onStartComend מחזיר). אם הסרוויס נכבל, זמן החייהפעיל שלו מסתיים כאשר ()unbind חוזר.