שקף 3:

במצגת זו אתן סקירה ראשונית של Gradle ולא אעבור באופן מעמיק על היכולות אלא יותר אסביר איך היא משתלבת בין הכלים הוותיקים וטיפה איך עובדים איתה.

שקף 5:

ANT הוא אחד כלי הבילד הראשון שפותח בעולם הג'אווה. הוא התחיל בסביבות שנת 2000 ומהר מאוד הפך לסטנדרט בתעשייה.

זהו כלי אימפרטיבי, כלומר אתה בעצם צריך להגיד לו בדיוק מה לעשות ואיך לעשות זאת.

כל קובץ בילד בנוי מטרגטים שכל אחד מהם מבצע פעולה מסוימת. בעצם אפשר לחשוב על כל טרגט כזה כפרוצדורה בעלת input ו output בקובץ הבילד. נראה זאת בהמשך.

שקף 6:

לפני שנראה קובץ לדוגמא, בו קודם נחשוב על מה הדרישות הבסיסיות מהבילד בעולם הג'אווה.

1. בג'אווה זה מאוד נפוץ להשתמש בספריות חיצוניות שלא חלק מהקוד שלנו, לכן עלינו להוריד ספריות אלה (שארוזות בקבצי JAR) לפני שנוכל לקמפל את הקוד שלנו.
2. קימפול הקוד תוך שימוש בתלויות
3. הרצת בדיקות לוודא תקינות
4. אריזת קבצי ה – class שנוצרו בשלב 2 לקובץ JAR יחיד לצורך הפצה.

עכשיו בו נראה איך הדרישות האלה מתורגמות לקובץ ANT לדוגמא.

שקף 7:

כל קובץ ANT בנוי מתג PROJECT יחיד ויכול להכיל PROPERTIES (משתנים) שיהיו בשימוש בקובץ הבילד. בדוגמא זו ניתן לראות שאנו מגדירים 3 משתנים:

1. SRC – איפה שנמצא הקוד שלנו.
2. BUILD – ספריה שתכיל את קבצי ה – CLASS שייווצרו בזמן הקימפול.
3. DIST – ספריה שתכיל את קובץ ה – JAR שיוצר בשלב האריזה.

עכשיו נראה את הטרגטים הרלוונטיים:

1. DIST - אחראי על יצירת קובץ ה – JAR (שלב 4 מהשקף הקודם). זהו הטרגט הראשון שיקרא, ובעצם ניתן לראות שהוא תלוי בטרגט נוסף בשם compile.
2. COMPILE – אחראי על קימפול הקוד שלנו. מקבל את מיקום קבצי ה – SRC כפרמטר ומייצר קבצי CLASS כ – OUTPUT. תלוי ב – CLEAN.
3. CLEAN – כשמו כן הוא. מקבל כ INPUT את ספריית ה – BUILD ודואג למחוק אותה על מנת להימנע משאריות מריצות קודמות.

שקף 8:

אם נחזור שוב לרשימת הדרישות נראה שבעצם ANT יכול לטפל בכל השלבים חוץ מהורדת התלויות (גם שלב 3 מטופל למרות שלא ראינו דוגמא). בעצם עלינו לוודא שכל התלויות יהיו זמינות על המחשב לפני הרצת הבילד.

ניהול התלויות והצורך לכתוב מחדש כל דרישה של מישהו מהבילד היו 2 החסרונות העיקריים של ANT והבסיס שעליו MAVEN נבנה. אז מזה בעצם MAVEN?

שקף 9:

MAVEN הוצג לראשונה בשנת 2004 תוך מטרה לפתור את הבעיות שהיו ב – ANT.

MAVEN הציג מספר דברים חדשניים כדוגמת:

1. Dependency management – ניהול תלויות אוטומטי. דומה ל – yum של REDHAT או Nuget של Microsoft.
2. Multi module – היכולת לפרק את הפרוייקט למודולים קטנים ולהגדיר תלויות בין מודולים אלה.

MAVEN הוא כלי דקלרטיבי, כאשר אתה רק מתאר את הפרויקט שלך ללא צורך בקביעת סדר הרצה או מה צריך לבצע ו – MAVEN עושה את כל השאר.

המודל של MAVEN מגדיר פעולות (lifecycle בשפה מקצועית), כאשר כל פעולה בנוייה מרשימה מוגדרת מראש של שלבים (PHASES). כל PHASE כזה ממומש בתורו ע"י PLUGIN. נראה זאת שוב בהמשך.

בנוסף MAVEN בא עם סט של קונוונציות משלו שמייתרות את הצורך להגדיר כל פעם מחדש דברים משותפים ויוצרים אחידות בין פרויקטים שונים. לדוגמא קבצי ה – SRC ישמרו תמיד תחת src\main\java וכ"ו. (convention over configuration).

שקף 10:

בשקף זה ניתן לראות את הארכיטקטורה של MAVE.

MAVEN משתמש בקובץ שנקרא POM.XML שבעצם מכיל את ההגדרות הדרושות ל – MAVEN בשביל לבנות את הקוד. במינימום הוא מכיל את השם של הפרוייקט והתלויות שלו.

בזמן ריצה MAVEN קורא את רשימת התלויות ומחפש אותם ברפוסיטורי הלוקאלי שלו (ספרייה שנשמרת על המחשב ומכילה את כל התלויות שנמצאו אי פעם ע"י MAVEN במחשב זה. במידה וזה לא קיים, הוא ניגש לאינטרנט ל – repository מרוחק ומוריד את התלויות לרפוסיטורי הלוקאלי לצורך שימוש בזמן הקומפילציה. פעם הבאה הקבצים יהיו זמינים כבר לוקאלית ופעולה זו תיחסך.

עכשיו בו נעבור בזריזות כל ה – featureים האחרים של MAVEN.

שקף 11:

ניהול תלויות. לצורך ניהול התלויות MAVEN הגדיר לכל מודול "כתובת" שבנויה מ – 3 שדות:

1. groupId
2. artifactId
3. version

השילוב של שלושתם מגדיר בעצם את הספרייה ב – repository שבו התוצר של המודול הזה יישמר.

שקף 12:

כמו שדיברנו מקודם, המודל של MAVEN מגדיר 3 פעולות ש – MAVEN יודע לעשות:

1. לבנות תוכנה (default lifecycle)
2. לנקות שאריות מריצה קודמת ( clean lifecycle)
3. בניית תיעוד (site lifecycle)

בעת הרצת MAVEN אנו אומרים ל – MAVEN איזה PHASE אנו רוצים להריץ, ולפי ה – lifecyce הוא מריץ את כל השלבים הרלוונטיים עד לשלב המבוקש כולל.

לדוגמא, הרצת MAVEN עם phase compile, יבצע:

Validate, initialize, generate-resources,…,compile

שקף 13:

בשקף זה ניתן לראות קובץ POM לדוגמא. כל קובץ כזה מכיל במינימום את ה "כתובת" של המודול שראינו מקודם ואת רשימת התלויות.

במקרה פה ניתן לראות שהמודול שלנו תלוי בספריה JUNIT וכמו שדיברנו, בזמן ריצה maven ידאג להוריד את הקבצים הרלוונטיים למחשב ולבנות את הקוד תוך שימוש בקבצים שהורדו אוטומטית.

שקף 14:

אם נחזור שניה לדרישות שהגדרנו מהבילד שלנו, נראה בעצם שכולם מטופלות ב – MAVEN.

MAVEN יודע לעשות כל מה שראינו ש ANT יודע ובנוסף גם לנהל לנו את התלויות.

עכשיו עולה השאלה בעצם, אז למה אנחנו צריכים את Gradle בכלל?

שקף 15:

בשביל לענות על כך אני רוצה קודם כל לספר לכם מה גרם לי להסתכל על Gradle מלכתחילה.

שקף 16:

אחת הסיבות שגרמו לי לעבור ל – GRADLE מלכתחילה היא התמיכה שלו ב – incremental build.

בעצם, המודל של MAVEN הוא מאוד נוקשה ומגדיר את ה – lifecycle וה – phases שאמורים להתבצע בזמן הבילד ללא יכולת לדלג על שום שלב (ניתן לזרז קומפילציה ע"י שימוש בקומפיילר חכם שיקמפל רק שינויים, אך זה לא יחסוך לנו את שלב הבדיקות או האריזה לדוגמא. דבר זה נתמך ב – Gradle.

בנוסף, לדעתי היום הדרישות שלנו מהבילד השתנו בגלל השנוי שנעשה בכלל בעולם התוכנה ודרישות אלה לא נענות ע"י מודל הרצה שפותח תוך שימוש בדרישות שהוגדרו לפני כעשור.

לדוגמא, היום יש לנו דברים כמו CD או מערכות שמשלבות קוד מכמה שפות שונות (לדוגמא java + JS), והדרישות שלנו מכלי הבילד השתנו ממה שהיו פעם.