**תרגול 13**

**Serialization:**

* סריאליזציה זהו מנגנון שממיר את המצב של אובייקטים לרצף כלשהו של בייטים שמייצגים את האובייקט
* דהסריאליזציה – התהליך ההפוך שלוקח רצף של בייטים ויוצר אובייקט אותו הוא מייצג
* מטרת התהליך היא לשמר מצב של אובייקט
* למה נחוץ?
  + לשחזר אובייקטים ששמרנו קודם
  + להעביר אובייקטים למישהו אחר (כדי שלא יצטרך להריץ מהתחלה)
  + גיבוי
* איך מבצעים?
  + המחלקה צריכה להצהיר שהיא מסכימה שיבצעו עליה סריאליזציה והיא עושה זאת ע"י מימוש ממשק Serializable שהוא marker interface (לא דורש מימוש של אף מתודה)
    - Marker interface – לא דורש לממש אף מתודה אלא מסמן לקומפיילר או לJVM שיש למחלקה איזושהי תכונה
* אם נשים לפני שדה את הmodifier הבא – transient – שדה זה לא יישמר בתהליך הסריאליזציה
  + נשתמש בכך כאשר לא נצטרך לשמור שדות כאלו או אם נרצה שתהליך הסריאליזציה יהיה מותאם
* התהליך:
  + ראשית ניצור OutputStream (ממשק) ונבחר ליצור FileOutputStream ונחשוב עליו כמשהו שכותב בייטים לאנשהו ובמקרה הספציפי הוא כותב לקובץ.
  + אותו ניתן כפרמטר לקונסטרקטור של ObjectOuputStream שאפשר לחשוב עליו בתור דקורטור או עטיפה – הוא מקבל משהו שאפשר לכתוב אליו והוא אומר שמה שנכתוב לשם הוא אובייקט.
  + נקרא על אובייקט הout שיצרנו למתודה writeObject() וניתן לו כפרמטר את האובייקט עליו נרצה להפעיל את התהליך
* תהליך הפוך:
  + אותו דבר עם input במקום output והוא יודע לקרוא אובייקטים.
  + בסוף נקרא ל-readObject()
* אם נכתוב כמה אובייקטים לקובץ אנו יכולים לבצע את אותו מספר קריאות על הקובץ וכל קריאה תקרא אובייקט אחר (לפי הסדר שהכנסנו)
* הקריאות האלו חייבות להיות בתוך try-catch כי יכולות לקרוא המון שגיאות בתהליך
* חוקי סריאליזציה:
  + בשביל לבצע סריאליזציה / דהסריאליזציה:
    - חייב לממש את הממשק Serializable
    - כל השדות שלו צריכים להיות Serializable או מוצהרים עם transient modifier
    - זה בגלל שהבדיקה למה אפשר לבצע עליו סריאליזציה קורת בזמן ריצה – הקומפיילר לא יכול לאשר אם אפשר לבצע את התהליך
  + כל הDATA MEMBERS, לא משנה אם פרטיים, מוגנים או פומביים, יעברו את תהליך הסריאליזציה – חוץ משדות סטטיים (לא מקושרים לאובייקט ספציפי) או שדות שהצהרנו עליהם עם המילה השמורה transient
  + writeObject() היא רקורסיבית (כלומר תהליך הסריאליזציה הוא רקורסיבי) – נבצע על האובייקט, ועל כל אחד מהשדות שלו, על כל אחד מהשדות שלהם וכן הלאה...
    - בעייתי במקרה שיש לנו לולאה – כמה אובייקטים שכל אחד שומר אובייקט של השני, אז התהליך הרקורסיבי לא יעצור
    - הפתרון – writeObject זוכרת דברים קודמים ששמרה ולא שומרת פעמיים אותו דבר. כלומר היא זוכרת דברים ששמרה קודם, ברגע שמגיעה למשהו שראתה כבר היא תשמור מצביע לשמירה הקודמת
      * כלומר גם אם שינינו ערך של אחד השדות של האובייקט עדיין הפונקציה מזהה שהיא כבר שמרה את האובייקט ותשמור רק מצביע למצב האחרון ששמרה (כלומר אם שמרנו מצב אחד של אובייקט גם אם שינינו את הערכים שלו לא נשמור אותו מחדש עד שלא נקרא את מה ששמרנו)
      * נוכל להשתמש ב-reset() שתאתחל ותמחק את כל מה שכבר שמרנו
        + נוכל גם לסגור ולפתוח את הSTREAM או לפתוח STREAM חדש אבל פחות יעיל.
* טיפוסים פרימיטיביים:
  + לא מבצעים עליהם סריאליזציה (אפשרי אבל לא נהוג)
  + במקום יש לObjectOutputStream מתודות עזר כמו writeInt
* שינוי מחלקה –
  + אם ביצענו כבר תהליך סריאליזציה ושמרנו אובייקט, ולאחר מכן שינינו את המחלקה וקימפלנו מחדש, אם ננסה לקרוא את האיבר יכולה לעלות בעיה שהמחלקה המעודכנת לא תצליח להעלות את האובייקט החדש.
  + יש שינויים שאפשר להתמודד איתם אבל לא עם כולם.
  + פתרון – שמירת גרסת מחלקה – נשמור שדה (מאפיין) סטטי בשם serialVersionUID
    - השדה הסטטי היחיד שנשמר בתהליך הסריאליזציה
    - אם השדה השתנה אז יש אי תאימות בין המחלקה ששמרנו לפיה את האובייקט לבין מצב המחלקה כרגע – במקרה כזה תזרק שגיאה של InvalidClassException
  + זה לא שדה חובה – הקומפיילר ייצר אותו אוטומטית אבל הוא ישנה את מספר הגרסה על שינויים קטנים שאנחנו לא היינו משנים גרסה בשבילם.
  + השדה חייב להיות סטטי ו-long
  + עדיף שיהיה פרטי
* סריאליזציה מותאמת:
  + אם נרצה לבצע סריאליזציה לאובייקט שיש לו שדות שהם ממחלקות שאינן serializable
  + נוכל להצהיר עליהן כtransient ואז לכתוב סריאליזציה מוגדרת אישית שתתייחס אליהן כמו שאנחנו נרצה כך שנוכל לשחזר אותם –
    - נדרוס את המתודה writeObject

Private void writeObject(ObjectOutputStream out) throws IOException{}

כותבת את שדות האובייקט הנתון לשדות האובייקט (מוגדרת בתוך המחלקה)

* + - ואת המתודה

Private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{}

מאתחלת את שדות האובייקט שהיא פועלת עליו לפי מה שכתוב בSTREAM

* + - אם הגדרנו את המתודות האלו הן מי שיקראו כאשר נבצע תהליך סריאליזציה ואם לא הגדרנו הקומפיילר ילך למימוש דיפולטיבי.
  + למעשה נקרא בתוכן לגרסא הדיפולטיבית – out.defaultWriteObject() ובדומה לread ואז נוסיף את השדות שנצטרך

**Reflection:**

* מנגנון שמאפשר לבחון מחלקות ואובייקטים שונים בזמן ריצה
* מוטיבציה –
  + שימוש במחלקות שמוגדרות ע"י המשתמש
  + כלי פיתוח (שיכולים לבחון את מה שאנו עושים)
  + כלי בדיקה ודיבאגרים
* מתי לא נשתמש בהם?
  + כמעט תמיד – אם אנחנו יודעים מה אנחנו מחפשים בזמן קומפילציה נעדיף לא להשתמש
  + לא נשתמש כדי להחליף ירושה או שימוש בממשקים!
* הטיפוסים הרלוונטיים:
  + Java.lang.Class – לכל מחלקה שנגדיר יש אובייקט CLASS רלוונטי אליה
  + Java.lang.reflect.\* - כל מיני דברים שמקושרים למחלקה – טיפוסים שמייצגים שדות, מתודות, קונסטרקטורים וכדומה
* תזכורת – שני סוגי טיפוסים
  + רפרנסים – מחלקות, ממשקים וכו
  + פרימיטיביים

REFLECTION מוגדרים רק לרפרנסים

* המחלקה Class:
  + לכל טיפוס רפרנס הJVM מייצר מופע של המחלקה CLASS שרלוונטי למחלקה שלנו
  + 3 דרכים להשיג אותו –
    - Class.foName(“<className>”)
    - Instance.getClass() – מחזיר את הטיפוס מחלקה של מופע מחלקה עליו נפעיל
    - <className>.class
  + שלושתם יחזרו כClass<>
  + ברגע שיש לנו אובייקט כזה של CLASS אפשר לעשות עליו המון דברים
    - למשל להשתמש ב-isInstance() עליו ולתת לו אובייקט
* המחלקה Method:
  + מייצג מתודה
  + נמצא ב- Java.lang.reflect.Method
  + שימושים:
    - METADATA על המתודה – שם המתודה, ערך החזרה, טיפוסי פרמטרים, modifiers וכו'.
      * Invoke מאפשרת להריץ את המתודה מחוץ למחלקה