**טיפים קטנים למבחן באופ (לא מוגה)**

* אפשר להגדיר מתודה כfinal. משמעות הדבר היא שלא ניתן לדרוס אותה. בעקבות כך, מתודה אבסטרקטית לא יכולה להיות final.
* Out זה שדה סטטי במחלקה System.
* פולימפורפיזם עובד רק בכיוון אחד (עלינו לשאול, האם ההשמה היא סוג של הreference).
* כל טיפוס שהוא iterbale אפשר לעשות עליו for.
* שלושת החלקים הנדרשים בסינגלטון הם: בנאי פרטי (שתומך בכך שלא נוכל לייצר את האובייקט מחוצה לה), משתנה סטטי יחיד שמחזיק אובייקט שדל המחלקה (זה הדרך שלנו ליצור את האובייקט), ומתודה סטטית שמחזירה את האובייקט (זה הדרך שלנו לגשת אל האובייקט מבחוץ
* אם נמצאים בחבילה אחרת, צריך גישה של public אם רוצים ליצור אובייקט אב חדש.
* מחלקה פנימית יכולה לגשת לשדות פרטיים של המחלקה החיצונית לה.
* מחלקה פנימית לא יכולה להגדיר מתודות סטטיות.
* מחלקה סטטית יכולה להגדיר מתודות שאינן סטטיות
* אפשר להכניס אובייקט בתור מפתח למילון, במימוש של equals וhashcode.
* לבדוק downCasting בזמן ריצה.
* משתנים בממשק תמיד מוגדרים static final.
* אפשר להגדיר משתנה final ולא להשים לו ערך, אם אתה נמצא במתודה.
* "הreference קובע מה מה מותר לרוץ, ההשמה קובעת מה ירוץ". יוצא מן הכלל הוא shadowing שבו הכלל הזה לא תקף."
* בהמשך לכך, Junit זה המקרה היחיד שראינו שבו חייבים את האנוטציה (@Test) נוטציות אחרות שראינו כמו @Override, @FunctionalInerface לא חייבים לכתוב.
* אם יש לך כמה catchים אז בשביל שלא תהיה שגיאת קומפליציה צריך שהכי נמוך בשרשרת ההיררכיה יהיה ראשון
* Class.forName מחפש את המחלקה. .getConstructors().newInstance() מוצא את הבנאים ומגדיר אובייקט חדש. כך ניתן לבדוק מחלקות בזמן ריצה
* לשים לב שלא תמיד אפשר לעבור עם for, ולכן עדיף להתנסות בiterator, hasnext() וnext (למשל מחיקת אובייקטים)
* לשים לב שלפעמים השגיאה היא לא ברורה מאליה. למשל – יכול להיות מצב בו אומרים שיש שגיאה בקוד, והכוונה להתנהלות לא צפויה של פונקציה לא מפורשת בקוד, או מחלקות שלא מממשות את comparable או iterable או שאין throws בחתימת הפונקציה.
* Unchecked לא צריך לבדוק ולכן אם יש אקספשיין לא בבלוק או בתנאי, אז מדובר בשגיאת זמן ריצה.
* במקרה בו רוצים לנסות ליצור דברים ללא יצירת דברים בגוף המחלקה, אפשר לממש מחלקות פנימיות בתוך מתודות.
* כאשר רוצים ליצור השוואה על פי מילון, עדיף ליצור שני מערכים באורכים שווים ולקבל את האינדקס המקסימלי של אחד מהם.
* כאשר בוצעה השמה – הwildcard לא רלוונטי.
* כאשר יש מתודה גנרית ומכניסים אליה משתנה מסוים, למשל מחרוזת, הקומפיילר מזהה את זה.
* כשיש משתנה גנרי שונה בין המתודה והמחלקה, זה דורס את המשתנה המקורי.
* הדרך הטובה ביותר לערוך פולימפוריפזם בין collections, זה עם wildcard
* בשביל ליצור כמה מתודות עם שם זהה: מספר שונה של ארגומנטים, טייפ שונה של ארגומנטים, סדר שונה של משתנים.
* @RunWith(Suite. class) ואז @Suite.SuiteClasses({.
* Equals צריך להיות סימטרי רפלקסיבי וטרנזיטיבי.
* Outer outer=new Outer(). Outer.Inner inner=outer.new Inner()

מבחנים שצריך לחזור:

**עקרונות חשובים בסיריאליזציה:**

* המחלקה המקורית צריכה לממש את Serializable.
* כל אחד מהשדות המורכבים צריך לממש את Serializable.
* ברגע שיצרנו אובייקט באמצעות תהליך הסיריאליזציה, הוא נשמר ולכן אין מצב של לולאה אינסופית (חבר שמכיר חבר וכו').
* מסיבה זו, כשאנחנו שומרים אובייקט, אנחנו שומרים מצביע לאובייקט, וגם אם נבצע קריאה לאובייקט מחדש, עדיין יישמר האובייקט המקורי. (הדרך לתקן את זה היא באמצעות reset או סגירת הstream).
* נשתמש ב-transient אם נרצה שלא יתבצע לשדה של אובייקט מסוים סיריאליזציה.
* נשתמש ב-SerialVerisonUID על מנת לשמור שינויים עקרוניים (אחרת, זה קורה באופן דיפולטיבי). ברגע שעושה תהליך דהסיריאליזציה, אז אם הID לא תואם, נזרק אקספשיין.
* על מנת לבצע פרקטית את הסיריאליזציה, עלינו:

1. ליצור קובץ שאליו נבצע את ההשמה. למשל:

OutputStream out=new FileOutputSteam("save.ser");.

1. להכניס (בתור decorator) את הקובץ לתוך המחלקה שמבצעת סיריאליזציה לאובייקט. למשל:

ObjectOutputStream oos=new ObjectOutpusStream(out).

1. לקחת את אובייקט הסיריאליזציה ולבצע עליו קריאה. למשל:

oos.writeObject(new Animal()).

* על מנת לבצע את הדהסיריאליזציה, עלינו לבצע אותו דבר (רק להחליף את ה Output בInput ואת הWrite ב-Read.)