JDBC :

JDBC הוא טכנולוגיה מבוססת Java גישה לנתונים (פלטפורמת Java-Standard Edition) מחברת Oracle. טכנולוגיה זו היא ה-API לשפת תכנות Java המגדירה כיצד לקוח יכול לגשת למסד נתונים. הוא מספק שיטות לביצוע שאילתות ועדכון נתונים במסד נתונים. JDBC מכוון למסדי נתונים יחסיים.

SQL:

SQL הינה שפה לטיפול ועיבוד מידע בבסיסי נתונים .תפקידה של השפה לאחזר נתונים בתשובה לשאילתות . מונחים בסיסיים :
Table,Attribut(Amuda),Domain(data of Attribute),Entry(Tuple)

SuperKey -אוסף כל השדות איתם ניתן לזהות רשומה . Primary Key -המפתח שבחרנו לטבלה.Foreign Key – שדה בטבלה שאינו מפתח אך מתייחס למפתח בטבלה אחרת. ניתן לבצע על טבלאות פעולות : איחזור ועדכון:Insert, Delete, Update,Modify

XML-eXtensible Markup Language:

שפה זו היא שפה לתיאור נתונים, תחביר גמיש לתיאור כל סוג של מידע,שפת תכנות לאחסון והעברה ב Web,כמו כן מאפשר לדפדפן להצע פעולות על הנתונים. ה- DTD מגדיר אילוצים על מבנה של מסמך XML ומהווה בעצם מפרט סינטקטי. Attributes הינו תג פותח היכול להכיל תכונות .מסמך הינו תקף אם הוא מכיל DTD ותואם לו. הגדרת תכונות ב-DTD נעשה על ידי ATTLIST .

DOM שומר את כל אלמנטים ה XML בזיכרון, בעוד ש SAX מפרסר אלמנט-אלמנט. מה שאומר ש DOM צורך יותר זיכרון אבל יש לו יותר יכולות (למשל פרסור מהסוף להתחלה) מאשר SAX.

SAX - קורא את המסמך סדרתית.DOM קורא את כל המסמך, ומכין עץ של המסמך בזיכרון לכן , אם יש מגבלת זיכרון עדיף להשתמש ב - SAX.
אם חשוב לך לעבור על המסמך הרבה פעמים קדימה/אחורה ולעשות מניפולציות עליו, עדיף DOM.

JDOM הוא API קוד פתוח, שתוכנן במיוחד עבור מתכנתי Java, המייצג את עץ XML כאלמנטים ותכונות. JDOM יכול לקיים אינטראקציה עם SAX או DOM. עם JDOM, לך לבנות מופע של בונה (org.jdom.input.SAXBuilder או org.jdom.input.DOMBuilder) ולאחר מכן להפעיל את המבנה (שיטה) על הקבלן לבנות אובייקט מסמך ממקור הקלט (קובץ , InputStream, כתובת, וכו ').

XPath, היא שפת שאילתה לבחירת צמתים ממסמך XML. בנוסף, XPath עוזרת בכך שניתן לחשב ערכים (לדוגמא, מחרוזות, מספרים או ערכים בוליאנית) מהתוכן של מסמך , שפת XPath מבוססת על ייצוג עץ של מסמך XML, ומספקת את היכולת לנווט סביב העץ, בחירת צמתים על ידי מגוון של קריטריונים.

Reflection:

Reflection הינו מנגנון לקבלת מידע על תוכן המחלקה: תכונות, שיטות, הרשאות וכד', מבלי שתהייה לנו גישה לקוד עצמו,מנגנון גנרי של שיקוף המידע,מנגנון זה מאפשר לנו להפעיל שיטות באופן דינאמי מבלי לדעת מראש מה יש בתוך המחלקה.ישנם מחלקות שבודקות תכונות למשל IsStatic ישנם מחלקות גישה למשל getMethods . מחלקות יצירה למשל constructor.newInstance . מחלקות קריאה invoke שיכולה להפעיל מתודות.

Client-Server:

הסרבר רק מגיב להודעות של הקליינט,הסרבר עובד מול קליינט יחידי, ה- server מופעל ומחכה שאפליקציות client יתחברו אליו.כאשר ה- server עולה יש לו כתובת שדרכה הוא מקבל מידע, client שרוצה להתחבר ל- server צריך לשלוח בקשה לכתובת זו . כתובת כזו מורכבת מ- 2 חלקים: כתובת IP, שהיא מספר המייצג מחשב ברשת מספר שהוא port עליו מאזין ה- server כדי שהסרבר ידע לעבוד מול כמה קליינטים בו"ז ע"י כך שיפתח ערוץ תקשורת יעודי לכל קליינט ויקשיב לו ב- thread נפרד. אפליקציית ה- server מופעלת ומחכה שאפליקציות client יתחברו אליה. עבור כל אפליקציית client המתחברת ל- server נוצר ערוץ תקשורת ייחודי (בעזרת thread).כלומר, client לא יכול לראות מהי התקשורת המועברת ל- server מ- client אחר.לאחר יצירת ערוץ התקשורת ה- client וה- server יכולים "לדבר".כאשר ה- client מתנתק ה- server סוגר את ערוץ התקשורת.ה- server הוא אפליקציה שאמורה לרוץ תמיד, ולכן לא תיזום ניתוק של client'ים המחוברים אליה. בעיה הנוצרת היא התקשורת בין השרת והלקוח היא סינכרונית, כלומר, על כל הודעה שהלקוח שולח לשרת הוא מצפה לקבל תשובה יתכן והלקוח ירצה לשלוח הודעה בכל רגע נתון, או שתי הודעות ברצף יתכן והסרבר ירצה ליזום הודעה לקליינט (כמו בדוגמא עם ה- broadcast) כלומר, יש לבצע שינוי במימוש של הקליינט כך שיהיה thread נפרד לשליחת הודעות לסרבר ו- thread נפרד המטפל בהודעות מהסרבר המימוש בקונסול הוא בעייתי מאחר והתצוגה תשתבש (יציג "אנא הכנס קלט" ופתאום תוצג הודעה מהסרבר) ולכן נממש ב- GUI.

נתאר שני תוכניות העוזרות לטפל בצד השרת של אפליקציית רשת (web) .

Servlet :

servlet זו תוכנית java המרחיבה את יכולותיהם של שרתים המארחים אפליקציות העובדות במודל של request-response. Multi-threaded מובנה. כל בקשה יוצרת thread חדש. (שקוף לנו) . קלט מהמשתמש אוטומטית מועבר למשתנה מסוג request. תוכנת השרת מריצה את ה servlet על הקלט שקיבלה מלקוח ומעבירה חזרה את תשובתו. שיטות get ו post הן שתי שיטות להעברת המידע. בשיטת get המידע מוצמד לסוף כתובת ה URL, הוא מוגבל באורכו וכמובן חשוף.לדוג'
http://localhost:8080/MyFirstServlet/GreetingServlet?firstName=igor&lastName=roch
בשיטת post אין הגבלה על כמות המידע והוא נשאר חסוי. ע"י response נקבל את ה output stream אליו נכתוב את התוצאה. ע"י request נוכל לגשת לפרמטרים השונים כפי שהוגדרו בטופס. נוכל לשמור מה שנרצה, לבצע כל חישוב שנרצה ולהחזיר תוצאה כאילו היינו כותבים לתוך קובץ. tomcat אחראי להעביר את התוצאה חזרה אל המשתמש דרך הדפדפן. Tomcat הינה תוכנית שרת הפועלת ברקע שאחראית להעברת הבקשות של המשתמש אל ה servlet המתאים ולהחזיר לו את התוצאה. Servlet הוא קוד JAVA בו משולב HTML .יש לקמפל את הקוד לפני .כמו כן הם רצים יותר מהר jsp .

JSP-Java Server Pages:

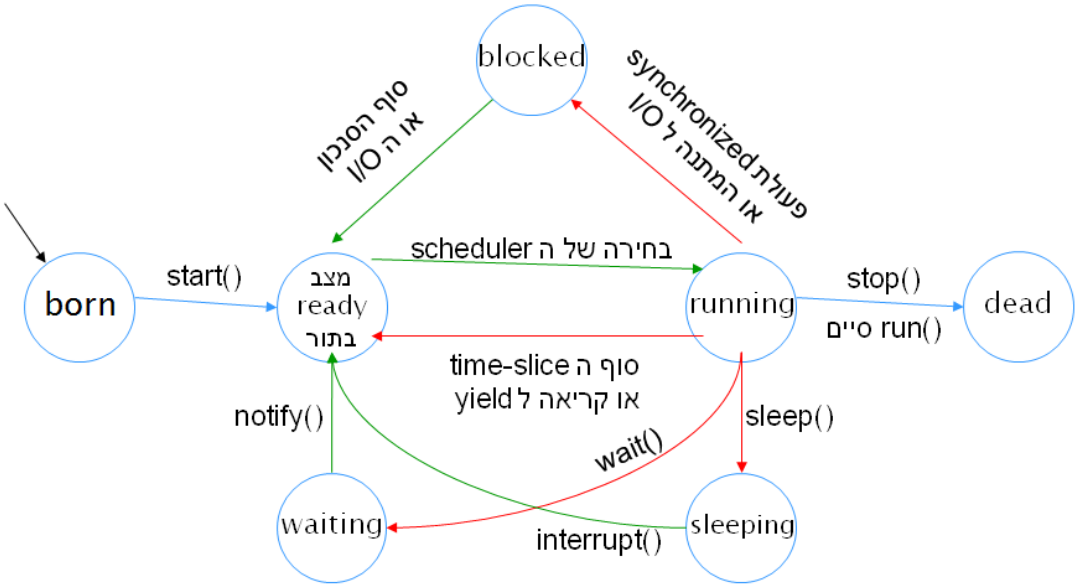
JavaServer Pages בראשי תיבות (JSP) :היא טכנולוגיית צד שרת מבוססת Java המאפשרת יצירה של דפיHTML, XML וקבצים אחרים, בתשובה לבקשות HTTP .טכנולוגיה זאת, מאפשרת לקוד Java ופעולות מוגדרות לשנות וליצור דפים בתשובה לכל בקשה. דף JSP גם יכול להיות דינאמי ע"י שילוב קוד java בתוכו, כאשר ההרצה נעשית בצד השרת. שאלה : מדוע כדאי להעביר מידע דווקא דרך request ולא בדרכים אחרות ? אלא מכיוון שאנו בסביבה מרובת threads – כמה threads שפועלים על אותו ה instance יכולים להפריע אחד לשני ולגרום לבעיות, הסנכרון אפשרי אך יכביד על פעולות המערכת. לכן כדאי להעביר מידע דרך האובייקטים request או session. הגיוני שהמשתמש יעבור דרך כמה דפים בהם תהיה לו אינטראקציה עם המערכת.session הוא אובייקט מידע הנוצר עבור כל משתמש, וניתן להכניס בו מידע ולחלץ ממנו מידע כבכל מבנה נתונים.JSP הינו קוד HTML שבו משולב JAVA .ב-JSP הקוד מתקפל בruntime . מאפשר לכתוב אפליקציות web ב-java.

Hibernate:

בתוכניות מורכבות המחזיקות מידע רב, נרצה להשתמש במסד נתונים ולא להתעסק ברמת מבנה הנתונים.נרצה אף להפריד בין שכבת מסד הנתונים לשכבת האפליקציה שעושה בו שימוש. (ויש לכך ארכיטקטורות שונות)נרצה שהקוד שלנו לא ישתנה אם החלטנו להחליף למסד נתונים חדש בעל שפת שאילתות אחרת.למשל בין mysql ל oracle. בעיה יותר חמורה היא שאנו נרצה לעבוד עם אובייקטים כחלק מתוכנית מונחית עצמים – ולא לעבוד עם טבלאות רלציוניות.האם SQL מכיר את המושג ירושה? פולימורפיזם? Hibernate היא ספריית Java למיפוי אובייקטי-רלציוני ,(object-relational mapping - ORM) המספקת תשתית למיפוי (התאמה) בין מודל התכנות מונחה-עצמים ,לבין בסיסי נתונים יחסיים מסורתיים. כאשר כותבים קוד העוסק בשימור נתונים ,(persistence) הייברנייט פותרת את בעיות חוסר ההתאמה בין מודל האובייקטים (תכנות מונחה-עצמים) למודל הרלציוני (בסיסי נתונים המורכבים מטבלאות) באמצעות החלפת הגישה הישירה לבסיס הנתונים ,בשימוש בפונקציות high-level המטפלות באובייקטים.השירות המרכזי ש Hibernate-מספקת הוא מיפוי בין מחלקות Java לטבלאות של בסיסי נתונים (ומיפוי בין טיפוסי הנתונים של Java לטיפוסי נתונים של .SQL) כמו כן Hibernate ,מספקת אמצעים לביצוע שאילתות ואחזור נתונים, והיא יודעת ליצור את קריאות ה SQL-בעצמה. Hibernate מנסה להקל על עבודת המתכנת בכך שהיא חוסכת ממנו את הצורך בטיפול ידני ב- result sets (המידע המוחזר מביצוע שאילתה על בסיס נתונים), וביצוע המרות בין אובייקטים. השימוש ב Hibernate- מאפשר ליישום להיות פורטבילי (יביל) בין כל בסיסי הנתונים מבוססי SQL הנתמכים על ידי הספרייה ,תוך תקורת ביצועים לא גבוהה . Hibernateהיא תוכנה חופשית .

:Threads

מערכת ההפעלה יכולה להריץ מספר תהליכים במקביל . כל אחד במרחב הכתובות שלו . בדרך כלל יש יותר תהליכים ממעבדים לכן המעבד צריך לשלב בין התהליכים ברמה שזה יראה כאילו הם רצים במקביל לשם כך ישנו מנגנון של Context Switch אבל היא יקרה מאוד. היתרון בלתכנת עם מספר תהליכים הינו שאם תהליך נתקע לא כל התוכנה קורסת .כעת, כל תהליך שרץ במערכת ההפעלה יכול להריץ כמה threads –תהליכונים.תהליכונים אלו הם חלקים שרצים "במקביל" כחלק מהתהליך עצמו ובמרחב הכתובות שלו.המשמעות שהתהליכונים שותפים לאותו מרחב כתובות היא שניתן בקלות להעביר מידע ביניהם (יותר בקלות מאשר בין תהליכים)כמו כן, המעבר בין תהליכון אחד לשני אינו יקר כמו context switch בין תהליך אחד לשני.החיסרון הוא כמובן שיש לנו הקצאות זמן עיבוד של תהליך אחד. בעצם למה צריכים Threads ? יש אלגוריתמים מקביליים,I/O זו פעולה שלוקחת המון זמן מנקודת המבט של המעבד לכן ניתן להריץ דברים אחרים בינתיים ,ביצוע משימות בלתי תלויות Timers,למיניהם,הכנה לסביבה מרובת מעבדים \ ליבות,עבודות שצריכות לרוץ ברקע, כמו gc.



deadlock נוצר כאשר שני threads ממתינים כל אחד שהשני "יעיר" אותו . בד"כ קורה כשכל אחד מחכה למשאב כלשהו מהשני... כדי להימנע ממקרה זה, יש לשמור על שני כללים פשוטים:החלטה על סדר הנעילות הגיוני ושחרור הנעילות בסדר הפוך.

ישנו מצב ששני threads עושים פעולה אנו מצפים לקבלת תשובה אך מקבלים תשובה שונה וזה מכיוון שהם התנגשו בזמן הפעולה ואחד שינה משהו והשני עבד עם הדבר שהראשון שינה .כדי למנוע זאת נשתמש במנגנון ה- synchronized .

במקרה שנגדיר public synchronized void run() כעת הthreads לא ירוצו במקביל אך תתקבלנה תשובה נכונה .

בס"ד

בעזרת ה' נעשה ונצליח !

סיכום תכנות מתקדם – תרגולים

<u>טיפים :</u>

1. **עבור assert אנחנו שמים בפנים בתוך הסוגריים מה שאנחנו רוצים שיתקיים ! כלומר הassert יעצור את התוכנית אם מה שכתוב בסוגריים אינו מתקיים !**
2. **שכתוב פונקציות const הכוונה דווקא לחתימה בסוף הפונקציה .**
3. **אם יש שאלת Templte צריך לפרט אם זה מקרה גנרי או ספציאליזציה.**
4. **בשאלה הראשונה של תיקון קוד הדברים הבאים אפשריים :**
   הזחות לא תקינות, תמיד לאתחל משתנה עם הצהרתו ,הגדרת משתנים ב-scope מינימאלי ,להחזיר ערכים מפונקציה ע"י מבנה , תמיד לשים סוגריים { } ( ) , אובייקט const תמיד בהשוואה יהיה בחלק השמאלי , לשים unsigned למשתנים => 0, אחרי מחיקת פוינטר להציב בו NULL (כך אם יהיה שוב delete התוכנית לא תיפול) , משתנים בצורה camelCase , מחלקות בצורה CamelCase , שמות קבועים לשדות למשל כולם עם _m , קונסטנטות תמיד UPPERCASE , לשנות warnings ל-error , עבודה עם משתנה שקיבלנו לפונ' במקום ען משתנה מקומי , סדר המשתנים שמקבלים לפונ' צריך להיות אחיד למשל קלט ואז כל הפלט ולא קלט,פלט ,קלט ,פלט . שם הפונ' צריך להעיד מה היא עושה, ביצוע גדול מדי של דברים בפונ' אחת . לכל if צריך else .
5. **assert מיועד לבדוק שאין באגים. שימוש ב-assert למימוש unit tests אינו מהווה נסיון למצוא באג ב-unit tests אלא בקוד שה-unit tests בודק. באופן כללי ה-assert משמש לבדיקת ההנחות שהנחתם בעת כתיבת קוד ומציאת המקומות בהן הן לא תואמות את המציאות. ב-unit tests הבדיקה היא לא של ההנחות שלכם אלא של המימוש שלכם.**
6. **Dependency injection – במקום שהפונ' מחליטה מאיפה לקחת נתונים אתה מחליט בשבילה ונותן לה את זה כפרמטר . למשל :**
   TimeOfDay t1 = TimeOfDay::now(FixedTimeGenerator(systime));
7. **ממשק תכנות יישומים) באנגלית ; Application Programming Interface :ראשי תיבות (API :הוא כינוי מקובל לערכות של ספריות קוד ,פקודות ,פונקציות ופרוצדורות מן המוכן, בהן יכולים המתכנתים לעשות שימוש פשוט, בלי להידרש לכתוב אותן בעצמם.**
8. **אין ירושה מרובה ב- JAVA לכן אם נרצה לעבוד עם Threads וכמו כן לרשת ממחלקה אחרת נעשה implements Runnable כי אז ירשנו רק ממחלקה אחת .**
9. **סיבות ש- Thread יעצור : Obj.wait , Sleeping , Stop(Finish) , Blocking(I/O) .**