עיעור אחרון

"הרצאה" מספר 15

נושאים שסטודנטים ביקשו

- Static Methods
- Factories
- Static vs. Dynamic calls
- Exceptions
  - Assert keyword
- Generics
- Factoring exercise, solution

שדות ושרותי מחלקה

```
public class Circle {
  // A class field
  public static final double PI= 3.14159;     // A useful constant

  // A class method: just compute a value based on the arguments
  public static double radiansToDegrees(double rads) {
    return rads * 180 / PI;
  }

  // An instance field
  public double r;                    // The radius of the circle

  // Two instance methods: they operate on the instance fields of an object
  public double area() {              // Compute the area of the circle
    return PI * r * r;
  }
  public double circumference() {     // Compute the circumference of the circle
    return 2 * PI * r;
  }
}
```

שדות ושירותי מחלקה

- שימוש בשירות מחלקה (מחוץ למחלקה)

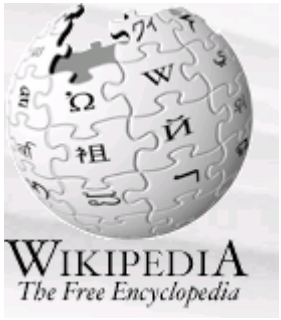
```
// How many degrees is 2.0 radians?

double d = Circle.radiansToDegrees(2.0);
```

- שימוש בשדה מופע (מחוץ למחלקה)

```
Circle c = new Circle(); // Create a Circle object; store a reference in c
c.r = 2.0; // Assign a value to its instance field r
Circle d = new Circle(); // Create a different Circle object
d.r = c.r * 2; // Make this one twice as big
```

- שימוש בשירות מופע (מחוץ למחלקה)

```
Circle c = new Circle(); // Create a Circle object; store in variable c
c.r = 2.0; // Set an instance field of the object
double a = c.area(); // Invoke an instance method of the object
```



בתי-חרושת - Factories

- תרגום חופשי מ-Wikipedia:

- בית חרושת הינו עצם המייצר עצמים אחרים. בית חרושת הינו הפשטה(Abstraction) של בנאי ושימושי לתבניות אתחול רבות (כמו Singletion).
- בד"כ לבית חרושת יש שירות ליצירת כל סוג עצם שהוא יכול לייצר. לעיתים ישנם פרמטרים לשירותים אלו אשר קובעים כיצד יוצר העצם החדש. שירותים אלו מחזירים מופע חדש
- בתי חרושת שמישים במיוחד במצבים בהם יצירת התייחסות לעצם מסוג מסוים יותר מסובכת מ"סתם" ליצור מופע חדש. בית החרושת יכול להחליט דינמית בדיוק איזה עצם ליצור, להחזיר מופע קיים ממאגר, לבצע אתחול מסובך וכו'

בתי-חרושת

- דוגמאות רלוונטיות שדיברנו עליהן
  - יצרנו מאגר של עצמים מסוג Version כדי לחסוך ביצירת מופעים חדשים
  - ניתן ליצור בית חרושת לייצור GameBoard כאשר סוג הלוח הנוצר תלוי בפרמטרים
- אם תחפשו, תמצאו בתי-חרושת רבים בג'אווה עצמה
  - SocketFactory
  - ThreadFactory
  - KeyFactory
  - …

Static vs. Dynamic Calls

- מהו Dynamic Dispatch?
  - זהו תהליך מיפוי משירות למימוש של השירות (קוד) בזמן ריצת התכנית
  - לא ניתן בזמן הידור (Compilation) לבצע את ההחלטה
- בג'אווה יש Dynamic Dispatch רק לגבי שירותי מופע

שיגור דינאמי Dynamic Dispatch

- ניתוח סטטי של קריאה target.method(…) קובע מהו הטיפוס ST של המשתנה target (בין אם שדה, משתנה מקומי או פרמטר): הטיפוס הסטטי (מנשק או מחלקה)
- בזמן ריצה, העצם ש-target מתייחס אליו הוא מטיפוס DT שיורש מ-ST. זהו הטיפוס הדינמי של target
- לכן רק בזמן ריצה ניתן לקבוע לאיזה שרות יש לקרוא: אם הוגדר שרות method(…) למחלקה DT, נקרא לו, אחרת יש לבדוק במחלקה ממנה ירש DT1, וכן הלאה
- מובטח שיימצא שרות (אחרת שגיאת קומפילציה)
- הקומפיילר יוצר קוד לביצוע הבחירה בזמן קבוע

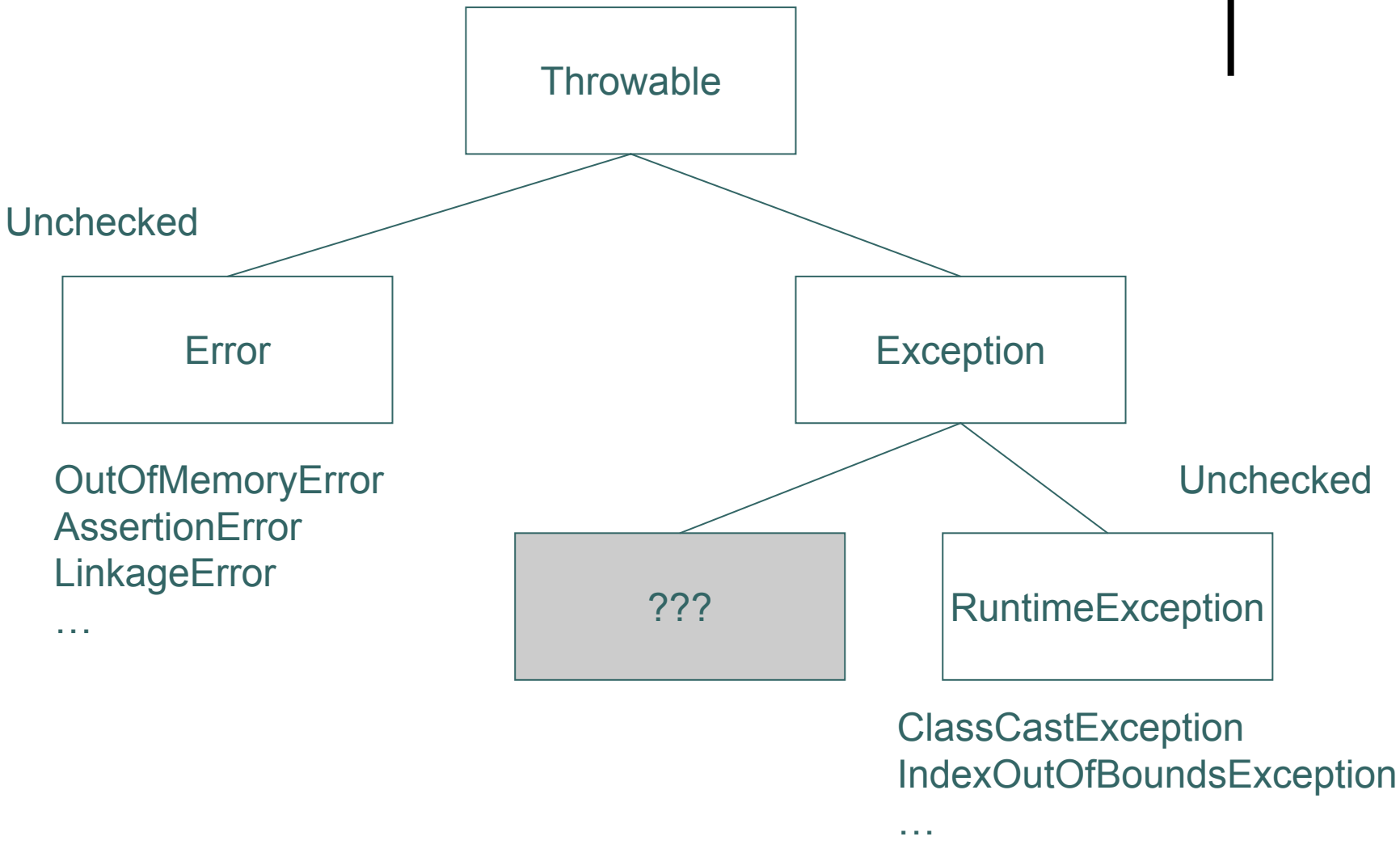
Static vs. Dynamic Calls

- מה קורה עם כל השאר? שדות מופע, שדות מחלקה, שירותי מחלקה
  - נקבעים על פי הטיפוס הסטטי
  - לא ניתן לעשות להם Override, רק להסתיר אותם

Class A
```
private int x;
```

Class B extends A
```
private int x;

private void f() {
   this.x = 10;
   super.x = 5;
}
```

חריגים


Throwable
Unchecked
Error
Exception
OutOfMemoryError
AssertionError
LinkageError
…
???
Unchecked
RuntimeException
ClassCastException
IndexOutOfBoundsException
…

חריגים

```
try {
    main code block
  } catch (Exception1 exc) {
    when Exception1 thrown inside try, transfer here
  } catch (Exception2 exc) {
    when Exception2 thrown inside try, transfer here
  } finally {
    optional part;
    Executed no matter how the try block is exited }
```

חריגים

- מתי נגדיר חריגים משלנו? למה זה טוב?
  - בתרגיל 5, המשתמש מכניס את המהלכים בתור מחרוזת
  - בספריית האלגברה לינארית שפיתחנו, מנסים להכפיל מטריצה 5X5 במטריצה 10X10
  - בשירות האמור לקרוא קובץ לעבד את הנתונים שבו תופסים חריג המציין כי הקובץ לא נמצא או שגוי

Assert

- פקודת ה-Assert נועדה לבדוק אינוריאנט, ז"א תנאי שתמיד אמת
- זהו כלי לתכניתן
- בריצה רגילה של התכנית, לא מורצות פקודות ה-Assert

Assert

```
if (x =  = 0) {
   ...
}
else {  // x is 1
   ...
}
```

```
if (x =  = 0) {
   ...
}
else {
   assert x =  = 1 : x  // x must be 0 or 1
   ...
}
```

```
switch(x) {
   case -1: return LESS;
   case 0: return EQUALS;
   case 1: return GREATER;
   default: assert false:x; // Throw AssertionError if x is not -1, 0, or 1.
}
```

נוודא שאף פעם לא נקבל ערך אחר

```
private static Object[] subArray(Object[] a, int x, int y) {
   assert x <= y : "subArray: x > y";   // Precondition: x must be <= y
   // Now go on to create and return a subarray of a...
}
```

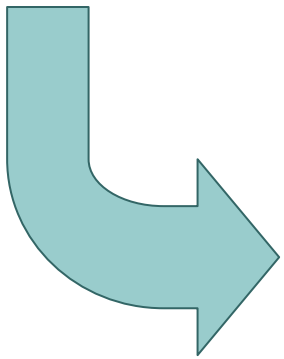
האם כל הפרמטרים נכונים

```
public void insert(Object o) {
   ...                  // Do the insertion here
   assert isSorted(  );  // Assert the class invariant here
}
```

האם האינוריאנט של המחלקה נשמר

Generics

```
public static void main(String[] args) {
    // This list is intended to hold only strings.
    // The compiler doesn't know that so we have to remember ourselves.
    List wordlist = new ArrayList();

    // Oops! We added a String[] instead of a String.
    // The compiler doesn't know that this is an error.
    wordlist.add(args);

    // Since List can hold arbitrary objects, the get() method returns
    // Object.  Since the list is intended to hold strings, we cast the
    // return value to String but get a ClassCastException because of
    // the error above.
    String word = (String)wordlist.get(0);
}
```



```
public static void main(String[] args) {
    // This list can only hold String objects
    List<String> wordlist = new ArrayList<String>();

    // args is a String[], not String, so the compiler won't let us do this
    wordlist.add(args);  // Compilation error!

    // We can do this, though.
    // Notice the use of the new for/in looping statement
    for(String arg : args) wordlist.add(arg);

    // No cast is required.  List<String>.get() returns a String.
    String word = wordlist.get(0);
}
```

Generics

```
public static void main(String[] args) {
    // A map from strings to their position in the args[] array
    Map<String,Integer> map = new HashMap<String,Integer>();

    // Note that we use autoboxing to wrap i in an Integer object.
    for(int i=0; i < args.length; i++) map.put(args[i], i);

    // Find the array index of a word.  Note no cast is required!
    Integer position = map.get("hello");

    // We can also rely on autounboxing to convert directly to an int,
    // but this throws a NullPointerException if the key does not exist
    // in the map
    int pos = map.get("world");
}
```

עוד דוגמה

וזה כבר
ממש להסתבך

```
public static void main(String[] args) {
    // A map from strings to their position in the args[] array
    Map<String,Integer> map = new HashMap<String,Integer>();

    // Note that we use autoboxing to wrap i in an Integer object.
    for(int i=0; i < args.length; i++) map.put(args[i], i);

    // Find the array index of a word.  Note no cast is required!
    Integer position = map.get("hello");

    // We can also rely on autounboxing to convert directly to an int,
    // but this throws a NullPointerException if the key does not exist
    // in the map
    int pos = map.get("world");
}
```

Generics

```
public static void printList(PrintWriter out, List<?> list) {
    for(int i=0, n=list.size(); i < n; i++) {
        if (i > 0) out.print(", ");
        Object o = list.get(i);
        out.print(o.toString());
    }
}
```

```
public static double sumList(List<?> list) {
    double total = 0.0;
    for(Object o : list) {
        Number n = (Number) o; // A cast is required and may fail
        total += n.doubleValue();
    }
    return total;
}
```

```
public static double sumList(List<? extends Number> list) {
    double total = 0.0;
    for(Number n : list) total += n.doubleValue();
    return total;
}
```

Generics

```
import java.io.Serializable;
import java.util.*;

public class Tree<V extends Serializable & Comparable<V>>
    implements Serializable, Comparable<Tree<V>>
{
    V value;
    List<Tree<V>> branches = new ArrayList<Tree<V>>();

    public Tree(V value) { this.value = value; }

    // Instance methods
    V getValue() { return value; }
    void setValue(V value) { this.value = value; }
    int getNumBranches() { return branches.size(); }
    Tree<V> getBranch(int n) { return branches.get(n); }
    void addBranch(Tree<V> branch) { branches.add(branch); }

    // This method is a nonrecursive implementation of Comparable<Tree<V>>
    // It only compares the value of this node and ignores branches.
    public int compareTo(Tree<V> that) {
        if (this.value == null && that.value == null) return 0;
        if (this.value == null) return -1;
        if (that.value == null) return 1;
        return this.value.compareTo(that.value);
    }

    // javac -Xlint warns us if we omit this field in a Serializable class
    private static final long serialVersionUID = 8335461436211334 67L;
}
```


נגדיר מחלקה ג'נרית בעצמנו

Generics

- אפשר לשכלל את העץ עוד יותר ולתת לתתי עצים להיות מטיפוסים של בנים של V

```
public class Tree<V> {
    // These fields hold the value and the branches
    V value;
    List<Tree<? extends V>> branches = new ArrayList<Tree<? extends V>>();

    // Here's a constructor
    public Tree(V value) { this.value = value; }

    // These are instance methods for manipulating value and branches
    V getValue() { return value; }
    void setValue(V value) { this.value = value; }
    int getNumBranches() { return branches.size(); }
    Tree<? extends V> getBranch(int n) { return branches.get(n); }
    void addBranch(Tree<? extends V> branch) { branches.add(branch); }
}
```

- ושימוש:

```
Tree<Number> t = new Tree<Number>(0);  // Note autoboxing
t.addBranch(new Tree<Integer>(1));     // int 1 autoboxed to Integer
```

Refactoring – Ex. 5

- רוצים ליצור תוכנית שמשחקת משחק **כלשהוא** אינטראקטיבית עם שחקן