תוכנה 1

תרגול 9 – שונות

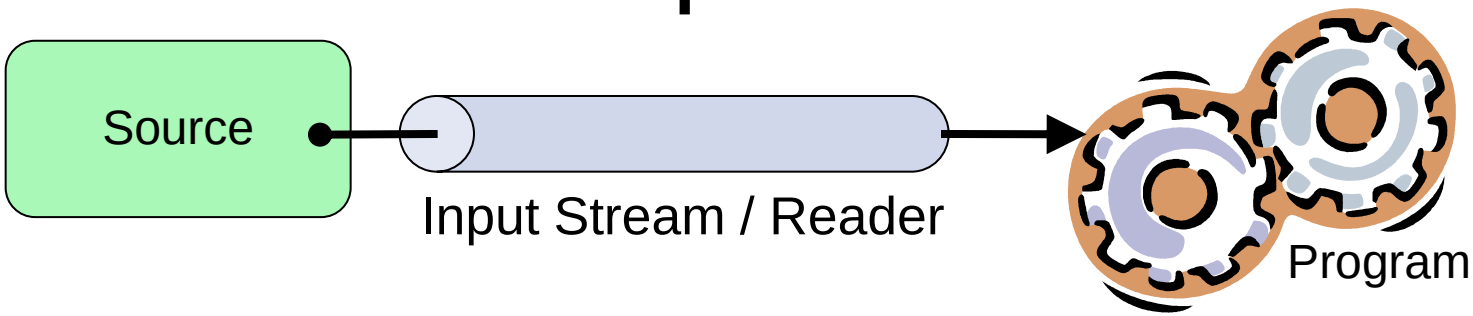
הדס צור ואסף זריצקי

Today

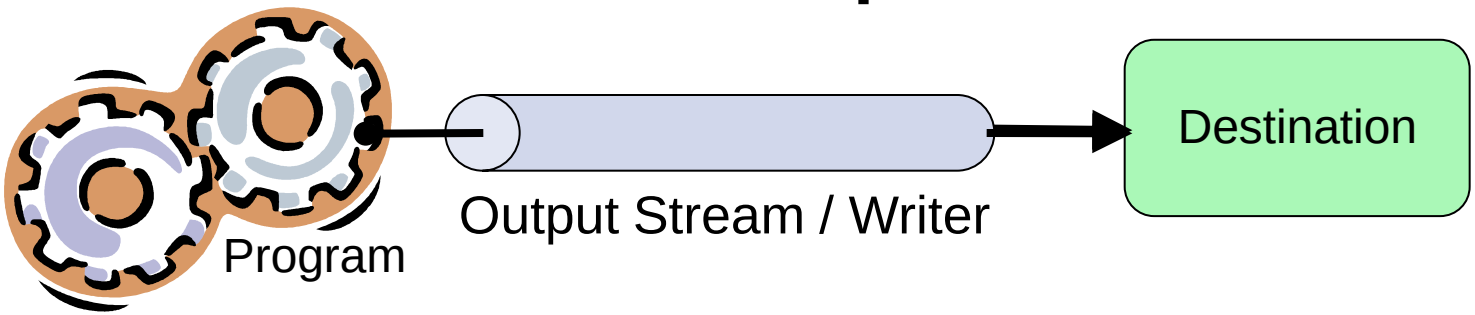
- More IO (object serialization)
- Equals / hashCode
- Static vs. Dynamic binding (maybe)

Streams

- A program that needs to read data from a source needs an **input stream** or **reader**



- A program that needs to write data to a destination needs an **output stream** or **writer**



Stream Wrappers

- Some streams wrap others streams and add new features.
- A wrapper stream accepts another stream in its constructor:

```
DataInputStream din =
    new DataInputStream(System.in);
double d = din.readDouble();
```

Stream Wrappers Example

- Reading a line of text from a file:

```
try {
    FileReader in =
        new FileReader("FileReaderDemo.java");

    BufferedReader bin = new BufferedReader(in);

    String text = bin.readLine();
    ...
} catch (IOException e) {...}
```

The File Class

- Represents pathname (file or directory)
- Retrieve meta data about a file
  - isFile / isDirectory
  - length
  - exists
  - …
- Performs basic file-system operations:
  - removes a file: delete()
  - creates a new directory: mkdir()
  - checks if the file is writable: canWrite()
  - …

Parsing

- Breaking text into a series of tokens
- The **Scanner** class is a simple text scanner which can parse primitive types and strings using regular expressions
- The source can be a stream or a string

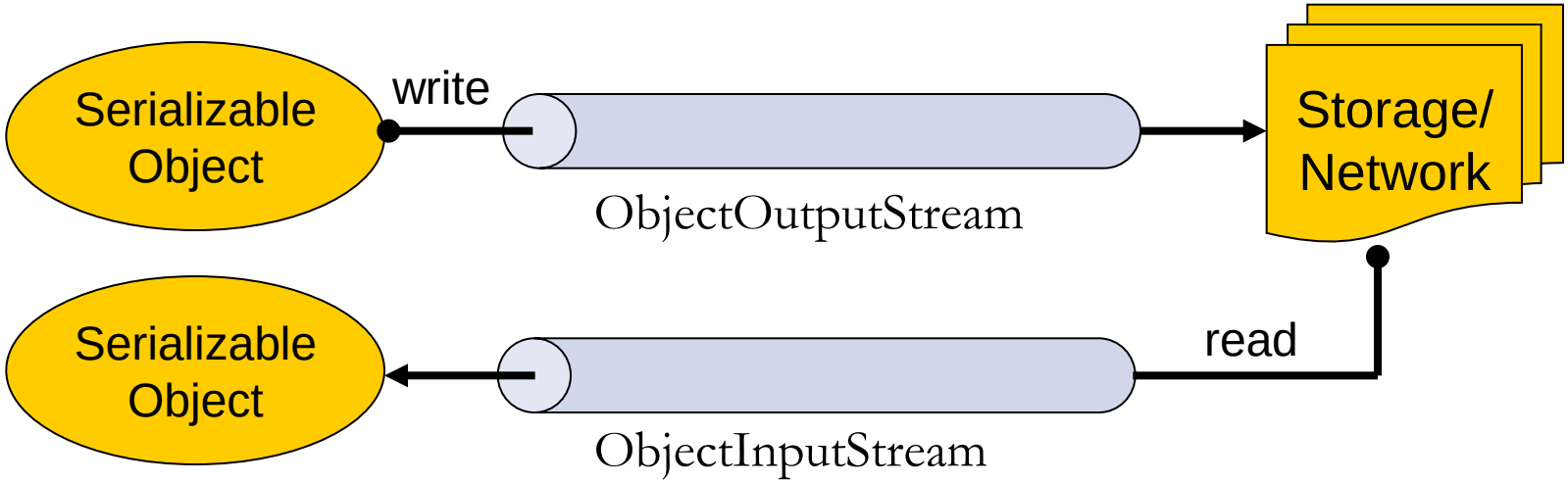
Object Serialization

- A mechanism that enable objects to be:
  - saved and restored from byte streams
  - persistent (outlive the current process)

- Useful for:
  - persistent storage
  - sending an object to a remote computer

The Default Mechanism

- The default mechanism includes:
  - The Serializable interface
  - The ObjectOutputStream
  - The ObjectInputStream



The Serializable Interface

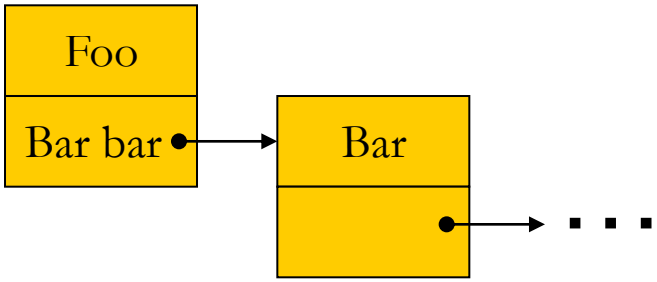
- Objects to be serialized must implement the java.io.Serializable interface
- An empty interface
- Some types are Serializable:
  - Primitives, Strings, GUI components etc.
- Subtypes of Serializable types are also Serializable

Recursive Serialization

- Can we serialize a Foo object?

```
public class Foo implements Serializable {
    private transient Bar bar;
    …
}

public class Bar implements Serializable {…}
```



- No, since Bar is not Serializable
- Solutions:
  1. Implement Bar as Serializable
  2. Mark the bar field of Foo as transient
  3. Customize the serialization process

HashMap Serialization

```
Map<Integer, String> map = new HashMap<…>();
...
ObjectOutputStream out = null;
try {
    out = new ObjectOutputStream(
                    new FileOutputStream("map.s"));
    out.writeObject(map);
} catch (IOException e) {
    ...
} finally  {
    ...
}
```

HashMap is Serializable, so are all the other **concrete** collection types we've seen

Reading Objects

```
ObjectInputStream in = null;
try {
    in = new ObjectInputStream(
             new FileInputStream("map.s"));
    Map<Integer, String> map =
        (Map<Integer, String>)in.readObject();
    System.out.println(map);
} catch (Exception e) {
    ...
} finally {
    ...
}
```

תזכורת: המחלקה Object

```
package java.lang;

public class Object {
    public final native Class<?> getClass();

    public native int hashCode();

    public boolean equals(Object obj) {
        return (this == obj);
    }

    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return getClass().getName() + "@" +
               Integer.toHexString(hashCode());
    }
    ...
}
```

מה יודפס?

```
public class Name {
    ...
    public static void main(String[] args) {
        Name name1 = new Name("Mickey", "Mouse");
        Name name2 = new Name("Mickey", "Mouse");
        System.out.println(name1.equals(name2));

        List<Name> names = new ArrayList<Name>();
        names.add(name1);
        System.out.println(names.contains(name2));
    }
}
```

false

false

הבעיה

- רצינו השוואה לפי תוכן אבל לא דרסנו את equals
  - מימוש ברירת המחדל הוא השוואה של מצביעים

```
public class Object {
    ...
    public boolean equals(Object obj) {
        return (this == obj);
    }
    ...
}
```

החוזה של equals

- **רפלקסיבי**
  - `x.equals(x)` יחזיר `true`
- **סימטרי**
  - `x.equals(y)` יחזיר `true` אמ"מ `y.equals(x)` יחזיר `true`
- **טרנזיטיבי**
  - אם `x.equals(y)` מחזיר `true` וגם `y.equals(z)` מחזיר `true` אז `x.equals(z)`
- **עקבי**
  - סדרת קריאות ל `x.equals(y)` תחזיר true (או false) באופן עקבי אם מידע שדרוש לצורך ההשוואה לא השתנה
- **השוואה ל null**
  - `x.equals(null)` תמיד תחזיר `false`

מתכון ל equals

```
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass()))
        return false;
    Name other = (Name) obj;
    return first.equals(other.first) &&
           last.equals(other.last);
}
```

1. ודאו כי הארגומנט איננו מצביע לאובייקט הנוכחי
2. ודאו כי הארגומנט איננו null
3. ודאו כי הארגומנט הוא מהטיפוס המתאים להשוואה
4. המירו את הארגומנט לטיפוס הנכון
5. לכל שדה "משמעותי", בידקו ששדה זה בארגומנט תואם לשדה באובייקט הנוכחי

טעות נפוצה

- להגדיר את הפונקציה equals כך:

```
public boolean equals(Name name) {
    return first.equals(other.first) &&
           last.equals(other.last);
}
```

- זו אינה דריסה (overriding) אלא העמסה (overloading)
- שימוש ב @Override יפתור את הבעיה

אז הכל בסדר?

```
public class Name {
    ...
    @Override public equals(Object obj) {
        ...
    }

    public static void main(String[] args) {
        Name name1 = new Name("Mickey", "Mouse");
        Name name2 = new Name("Mickey", "Mouse");
        System.out.println(name1.equals(name2));

        List<Name> names = new ArrayList<Name>();
        names.add(name1);
        System.out.println(names.contains(name2));
    }
}
```

יודפס true

יודפס true

כמעט

```
public class Name {
    ...
    @Override public equals(Object obj) {
        ...
    }

    public static void main(String[] args) {
        Name name1 = new Name("Mickey", "Mouse");
        Name name2 = new Name("Mickey", "Mouse");
        System.out.println(name1.equals(name2));

        Set<Name> names = new HashSet<Name>();
        names.add(name1);
        System.out.println(names.contains(name2));
    }
}
```

יודפס true

יודפס false

hashCode ו equals

חובה לדרוס את hashCode בכל מחלקה
שדורסת את equals!

החוזה של hashCode

- **עקביות**
  - מחזירה אותו ערך עבור כל הקריאות באותה ריצה, אלא אם השתנה מידע שבשימוש בהשוואת **equals** של המחלקה
- **שוויון**
  - אם שני אובייקטים שווים לפי הגדרת equals אזי hashCode תחזיר ערך זהה עבורם
- **חוסר שוויון**
  - אם שני אובייקטים אינם שווים לפי equals לא מובטח ש hashCode תחזיר ערכים שונים
  - החזרת ערכים שונים יכולה לשפר ביצועים של מבני נתונים המבוססים על hashing (לדוגמא, HashSet ו HashMap)

מימוש hashCode

```
@Override public int hashCode() {
    return 31 * first.hashCode() + last.hashCode();
}
```

- השתדלו לייצר hash כך שלאובייקטים שונים יהיה ערך hash שונה
- המימוש החוקי הגרוע ביותר (לעולם לא לממש כך!)

```
@Override public int hashCode() {
    return 42;
}
```

תמיכה באקליפס

- אקליפס תומך ביצירה אוטומטית (ומשולבת) של hashCode ו equals
  - בתפריט Source ניתן למצוא Generate hashCode() and equals()

Static versus run-time binding

```
public class Account {
   public String getName(){...};
   public void deposit(int amount) {...};
 }

 public class SavingsAccount extends Account {
   public void deposit(int amount) {...};
 }
```

```
Account obj = new Account();
obj.getName();
obj.deposit(…);

obj = new SavingsAccount();
obj.getName();
obj.deposit(…);
```

Static binding (or early binding)

- Static binding: bind at compilation time
- Performed if the compiler can resolve the binding at compile time
  - Static functions
  - Access to member variables
  - Private methods
  - Final methods

Static binding example

```
public class A {
   public String someString = "member of A";
}
public class B extends A {
   public String someString = "member of B";
}

A a = new A();
A b = new B();
B c = new B();
System.out.println(a.someString);
System.out.println(b.someString);
System.out.println(c.someString);

Output:
   member of A
   member of A
   member of B
```

When to bind?

```
void func (Account obj) {
    obj.deposit();
}
```

- What should the compiler do here?
  - The compiler doesn't know which concrete object type is referenced by `obj`
  - the method to call can only be known at run time (*because of polymorphism*)
  - <u>Run-time binding</u>