Programming Mobile Devices

Record Storage + Serialization

University of Innsbruck

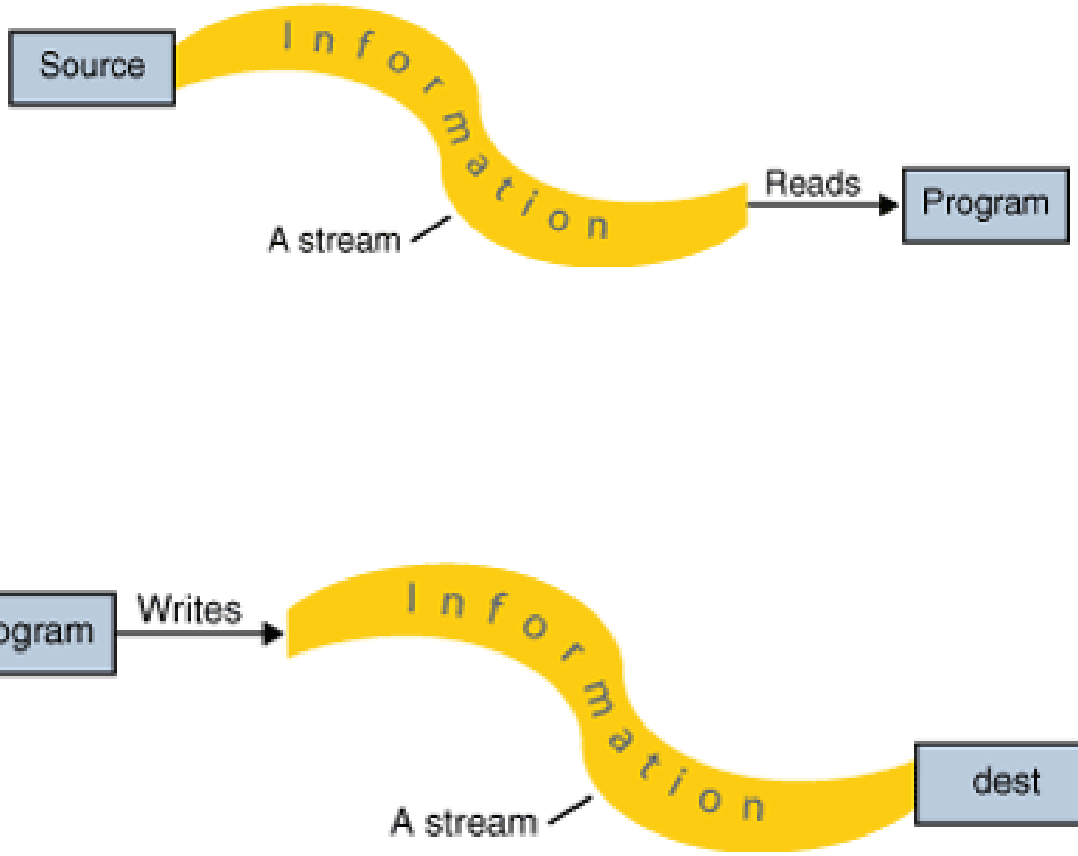
WS 2009/2010

STI · INNSBRUCK

thomas.strang@sti2.at

The J2SE I/O System

- Any program consists of three parts: input, process and output



Reading and Writing to/from Streams

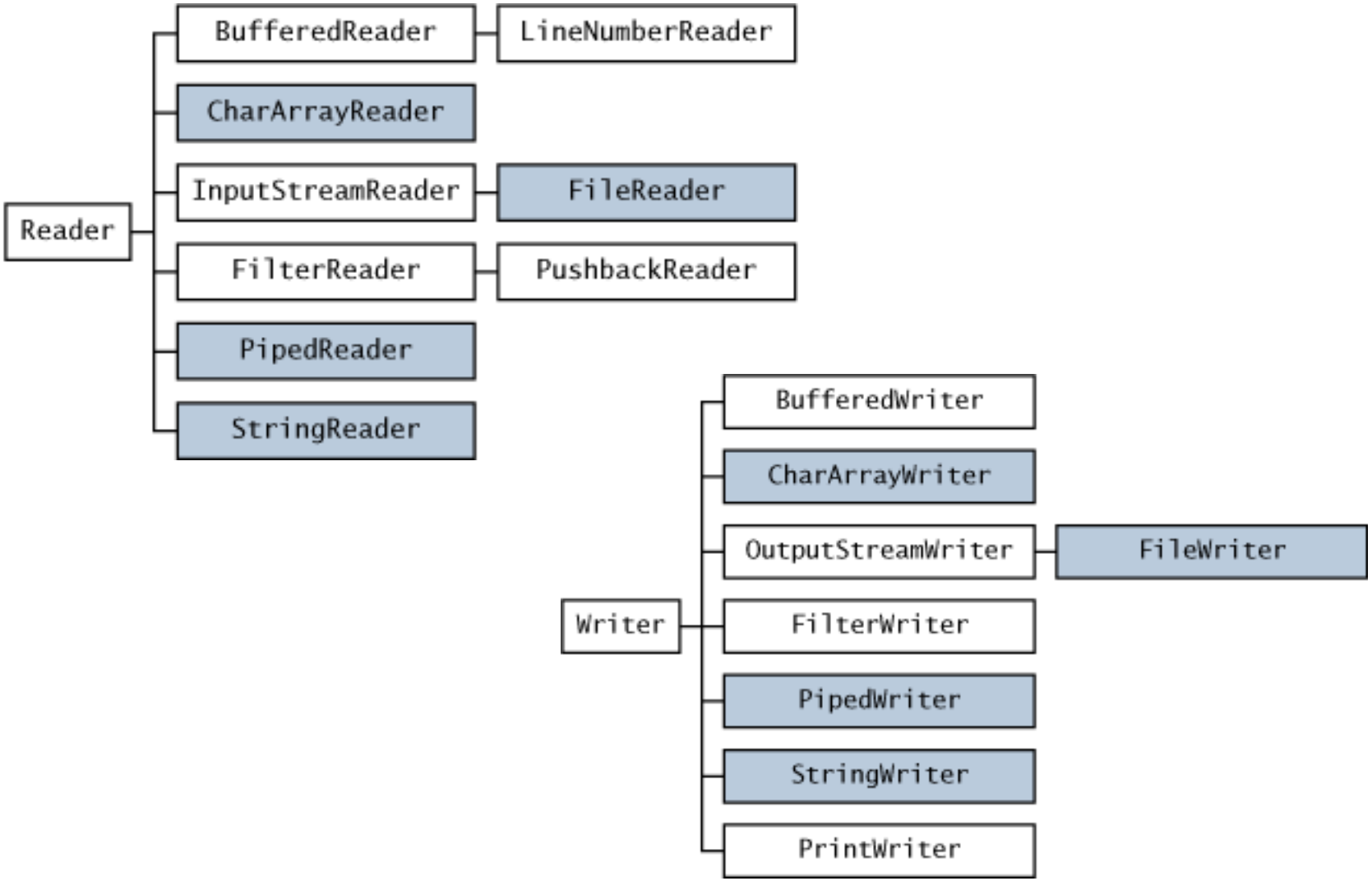
Reading

- open a stream
- while more information:
  read information
- close the stream

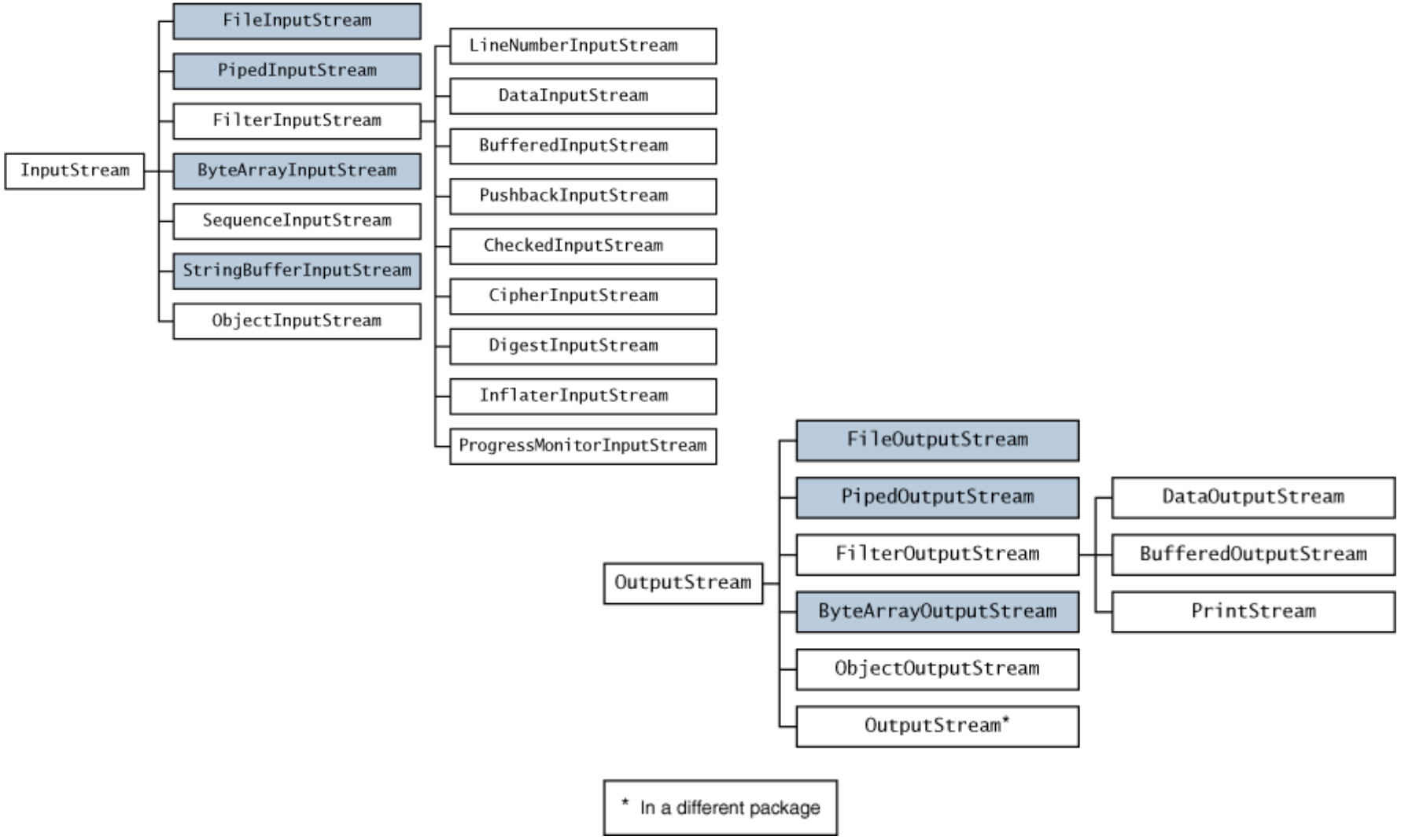
Writing

- open a stream
- while more information:
  write information
- close the stream

Character Streams



Byte Streams



Examples – Copy a file

```
import java.io.*;

public class Copy {
    public static void main(String[] args) throws IOException {
        File inputFile = new File("original.txt");
        File outputFile = new File("copy.txt");

         FileReader in = new FileReader(inputFile);
         FileWriter out = new FileWriter(outputFile);
         int c;

         while ((c = in.read()) != -1)
            out.write(c);

         in.close();
         out.close();
    }
}
```

Examples – Chaining streams

```
BufferedReader in
   = new BufferedReader(new FileReader("test.txt"));
// read more than necessary from file when read() is performed
// and thus increase performance on subsequent read()'s



PrintWriter out
   = new PrintWriter(new BufferedWriter(new FileWriter("test.out")));
```

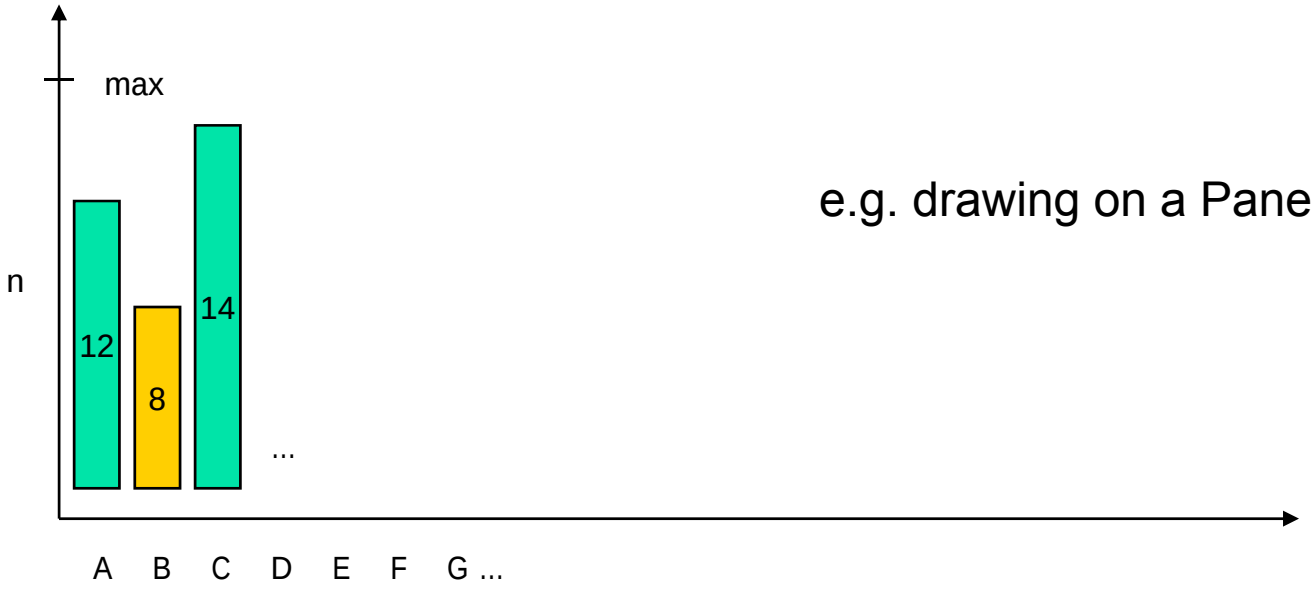
Examples – BufferedReader in Action

```
import java.io.*;
class BufferedReaderTest  {
   public static void main (String args[]) {
        String thisLine;
        //Loop across the arguments
        for (int i=0; i < args.length; i++) {
          try {   //Open the file at i for reading and read
                BufferedReader br =
                        new BufferedReader(new FileReader(args[i]));
                while ((thisLine = br.readLine()) != null) {
                        System.out.println(thisLine);
                }
          }
          catch (IOException e) {
                System.err.println("Error: " + e);
          }
        }
   }}
```

Exercise 6 – from streams to display

Write an AWT or SWING application, which allows the user to select a URL, reads the content and shows a histrogram (visual distribution) of characters (ignore upper/lower case).

Example: read from http://www.o-bible.org/download/bbe.txt and show the following



e.g. drawing on a Panel in a Frame

Persistent Storage in J2ME

Mobile Phones usually don't have harddisks

(Exception: Nokia N91 

)

Persistent Storage

- Instead of Harddisks, they usually have Flash Memory
  - SecureDigital (SD cards)
  - Compact Flash (CF cards)
  - MemoryStick
  - etc.

- Characteristics of Flash Memory:
  - capacity: up to a few GB
  - sufficiently fast in reading data
  - **slow in (re-)writing data**

File System?

- even if x GB, maybe just 64 kB for apps
- installation: filesystem not visible
- from MIDP: just DB-like storage

MIDP: Record Management System (RMS)

- RMS is modelled after a simple record oriented DB
- Core compoment is the `RecordStore` class
- Each RecordStore is associated with the MIDlet suite
  - If a suite is removed, the aligned RecordStore is also removed
  - maybe multiple RecordStores within one suite, but names must be unique within that suite
  - access to a RecordStore limited to the MIDlets within the same suite (MIDP 1.0)

Records

- Each `Record` in a `RecordStore` is an array of bytes
- Records are uniquely identified by their `recordId` (which are set only by the RMS!), an integer starting with 1

Example

- within a MIDlet you create a new RecordStore

"filename"

```
RecordStore rs = null;
try {
    rs = RecordStore.openRecordStore("myStore", true);
}
catch (RecordStoreException e) {}
```

- write a few records

every record is a byte array

```
try {
    String rec = "Firstname, Lastname, Age";
    byte[] ba = rec.getBytes();
    int recId = rs.addRecord(ba, 0, ba.length);
}
catch (RecordStoreException e) {}
```

- finally close the RecordStore before leaving MIDlet

```
try {
    rs.closeRecordStore();
    rs = null;
}
catch (RecordStoreException e) {}
```

Stream I/O of MIDP/CLDC is similar (≈ subset)

Stream I/O with RMS

- write to RMS through a stream

```
try {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream( baos );
    dos.writeInt( 4711 );
    byte[] ba = baos.toByteArray();
    recId = rs.addRecord(ba, 0, ba.length);
}
catch (Exception e) {}
```

- read from RMS through a stream

```
try {
    byte[] ba = new byte[ rs.getRecordSize( recId )];
    rs.getRecord(recId, ba, 0);
    ByteArrayInputStream bais = new ByteArrayInputStream( ba );
    DataInputStream dis = new DataInputStream( bais );
    int value = dis.readInt();
}
catch (Exception e) {}
```

Iterating through a RecordStore

- Using the `RecordEnumeration` interface

```
try {
    RecordEnumeration re = rs.enumerateRecords(null, null, false);
    while ( re.hasNextElement() ) {
        byte[] ba = re.nextRecord();
        // assuming records are strings
        System.out.println( new String(ba) );
    }
}
catch (Exception e) {}
```

Searching in RMS… is an extension of Iterating

- Function `match` must be implemented …

```
public class MyFilter implements RecordFilter {

    public boolean matches(byte[] candidate) {
        String c = new String(candidate);
        if (c.startsWith("September"))
            return true;
        return false;
    }
}
```

- … and provided to the `RecordEnumeration`

```
try {
    RecordEnumeration re = rs.enumerateRecords(new MyFilter, null, false);
    while ( re.hasNextElement() ) {
        byte[] ba = re.nextRecord();
        // assuming records are strings
        System.out.println( new String(ba) );
    }
}
catch (Exception e) {}
```

Order of the Records

- Function `compare` must be implemented and…

```
public class MyComparator implements RecordComparator {

    public int compare(byte recA[], byte recB[]) {

        String nameA = new String( recA );
        String nameB = new String( recB );
        int num = nameA.compareTo( nameB );

        if (num > 0)      return RecordComparator.FOLLOWS;
        else if (num < 0) return RecordComparator.PRECEDES;
        else              return RecordComparator.EQUIVALENT;
    }
}
```

- … provided to the `RecordEnumeration`

```
try {
    RecordEnumeration re =
        rs.enumerateRecords(new MyFilter, new MyComparator, false);
    while ( re.hasNextElement() ) {
        byte[] ba = re.nextRecord();
        System.out.println( new String(ba) );
    }
}
```

Inter-Suite RMS Access

- MIDP 2.0 allows for explicit sharing of record stores if the MIDlet creating the RecordStore chooses to give such permission
  - unique naming now requires to include the suite name
  - one of two modes is defined when RecordStore is *created*:
    - AUTHMODE_PRIVATE – as in MIDP 1.0 (default)
    - AUTHMODE_ANY – RecordStore may be accessed from other suites as well, if the other suite can name the RS

Exercise 7 – a virtual highscore saving

- store your name, age and latest score
  - String lastname
  - String firstname
  - int age
  - int highscore
- exit app + start app
- reload and display data

Exercise 8 – combine 6 & 7

- Implement a graphical histogram function (counting letters) of exercise 6 on MIDP emulator
- Count the letters of your stored data (exercise 7)
- Show it to me!

Serialization/Deserialization

- Equally used terms
  - marshalling/unmarshalling
- Why serialization?
  - objects in use
    - main memory
  - objects in storage
    - RMS
  - objects in transit
    - on the wire
    - in the air

Simple Serialization

- There is no `java.io.Serializable` in J2ME
- Every object has a `toString()` method, which is a very simple form of serialization
- For more complex classes (structs), this may not be sufficient and/or error-prone, in particular for the de-serialization step

Self-describing Serialization

- The idea of *self-describing serialization* is to encode information about the class where the object is an instance of, versioning information, length in byte array representation etc.

- Example:

```
public class Entry
{
    private String id = "";
    private String city = "";
    private String name = "";
    private String description = "";
    // … more attributes
    // … some member functions
}
```

Example (cont'd): Serialization

```
public byte[] toByteArray() {

    // serialize this instance into byte array
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream( baos );
    try {
        dos.writeUTF( this.getClass().getName() );
        dos.writeUTF( this.getId() );
        dos.writeUTF( this.getCity() );
        dos.writeUTF( this.getName() );
        dos.writeUTF( this.getDescription() );
        // … continue on other attributes
    }
}
```

Example (cont'd): De-Serialization

```
// static de-serialization function
static public Entry fromByteArray(byte[] ba) {

    ByteArrayInputStream bais = new ByteArrayInputStream( ba );
    DataInputStream dis = new DataInputStream( bais );
    return new Entry( dis );
}

// constructor
protected Entry(DataInputStream dis) {

    try {
        String classname = dis.readUTF();
        if (!classname.equals(this.getClass().getName()))  // check!
            throw new RuntimeException(
                "Cannot deserialize " + this.getClass().getName() +
                " from " + classname);

        setId( dis.readUTF() );
        setCity( dis.readUTF() );
        setName( dis.readUTF() );
        setDescription( dis.readUTF() );
        // … continue with other attributes
    }
}
```

Using Inheritance: Serialization

```
public class Note extends Entry {

    private String geoNote = "We are in Innsbruck";

    // serialize to byte array
    public byte[] toByteArray() {

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream( baos );
        try
        {
            // first: our ancestors
            dos.write( super.toByteArray() );

            // second: who we are
            dos.writeUTF( this.getClass().getName() );

            // third: what makes us more special than an Entry
            dos.writeUTF( geoNote );
        }
        // … and so on
```

Using Inheritance: De-Serialization

```
// static de-serialization function
static public Note fromByteArray(byte[] ba) {

    ByteArrayInputStream bais = new ByteArrayInputStream( ba );
    DataInputStream dis = new DataInputStream( bais );
    return new Note( dis );
}

// constructor
protected Note(DataInputStream dis) {

    try {
        // first: construct our ancestor
        super(dis);

        // second: check if we are right here
        String classname = dis.readUTF();
        if (!classname.equals(this.getClass().getName()))  // check!
            throw new RuntimeException(
                "Cannot deserialize " + this.getClass().getName() +
                " from " + classname);

        // third: de-serialize the rest here
        setGeoNote( dis.readUTF() );
    }
}
```

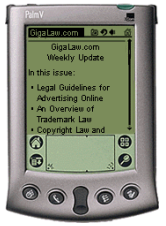
Exercise 9 – serialize/deserialize to RMS

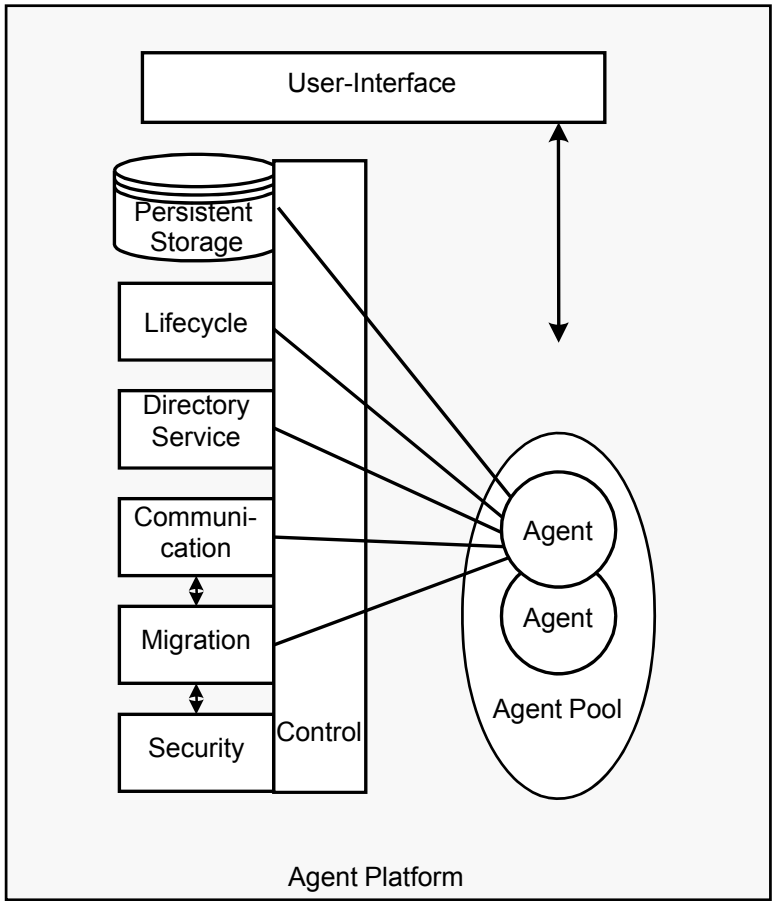
- Create 3 classes
  - abstract Shoppinglist
    - attribute String whereToShop
  - ShoppinglistMan extends Shoppinglist
    - attribute int socksSize
    - attribute int socksAmount
  - ShoppinglistWoman extends Shoppinglist
    - attribute String lipstickColor
    - attribute int lipstickAmount
- create a few (>5) instances of each and store them in random order to RMS
- quit/restart app, load and show shoppinglist on screen

Another Example: Agents

Agent Platform on Small Mobile Devices – does it make sense?

Some of the general problems





- No or reduced file system (e.g. simple record storage)
- Limited number of simultaneously running agent instances / connections
- Interferences between agents due to limited processing power
- Inter-platform communication only when connected
- overhead agent vs. program

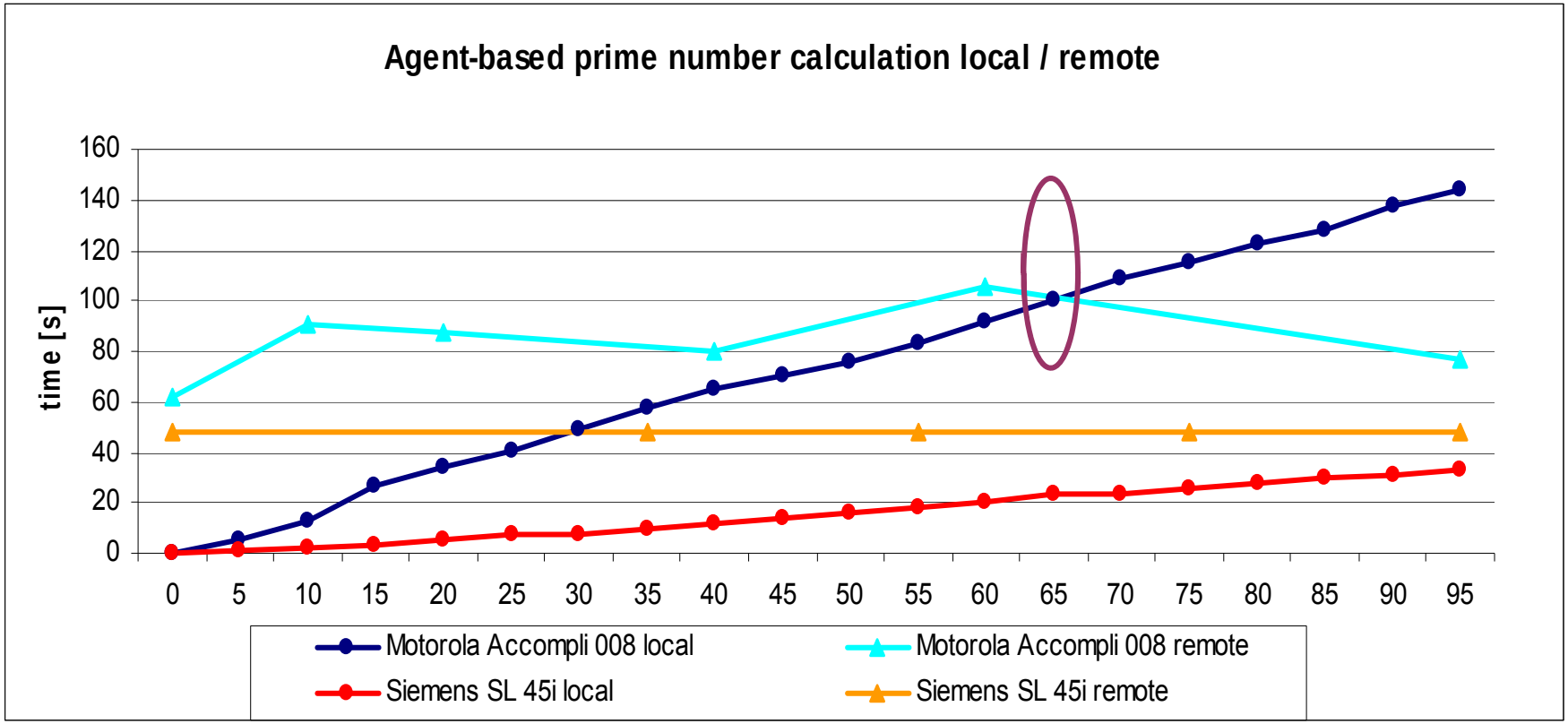
Pro's and Con's

+ using local computing power (vs. thin client approach) enables network situation sensitive work, up to autonomy when not connected
+ private data is often not required to be transmitted
+ enables cost-conscious bandwidth allocation

- overhead of agents and platform itself vs. 'normal' programs, in particular where resources are strongly limited
- platform independance of language may not be given (in particular for mobile agents)

Exact analysis of individual task is mandatory

Performance Example

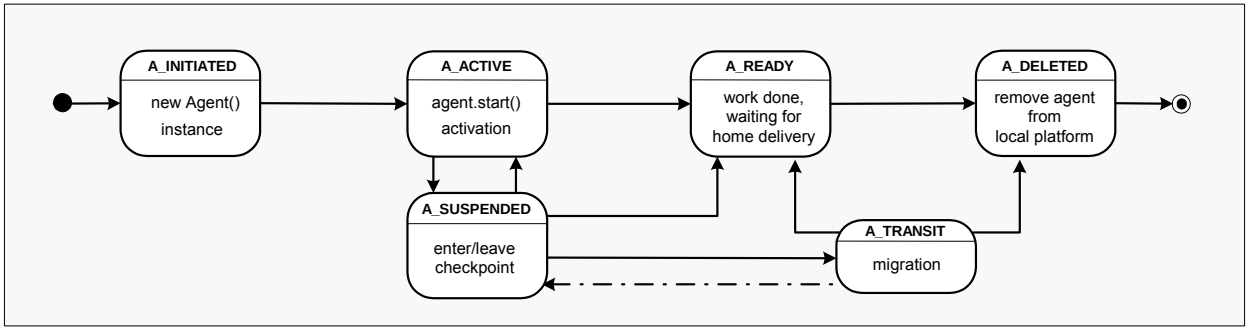


[Strang & Meyer, 2002]

- "unusual" performance values
- knowledge about algorithm- and device-specific performance parameters can be used for agent management

Weak Migration due to Closed Late Binding

State diagram



Migration from Mobile Device (each agent)

1. Type negotiation
2. Serialization
3. Transmit "spirit" of agent
4. Transit state handling

Migration to Mobile Device (MIDP 1.0: poll only)

1. Type negotiation
2. Check number of agents to transmit
3. Transmit "spirit" of agents
4. Deserialization and re-activation

Timings