Overview

A. Basic machinery

execution model, byte code vs. native code, language aspects

B. Memory management

basic schemes, memory types: transient vs. persistent, garbage collection

C. Atomicity and transactions

basic schemes, system-level vs. user-level transactions

D. OO programming w/ resource constraints

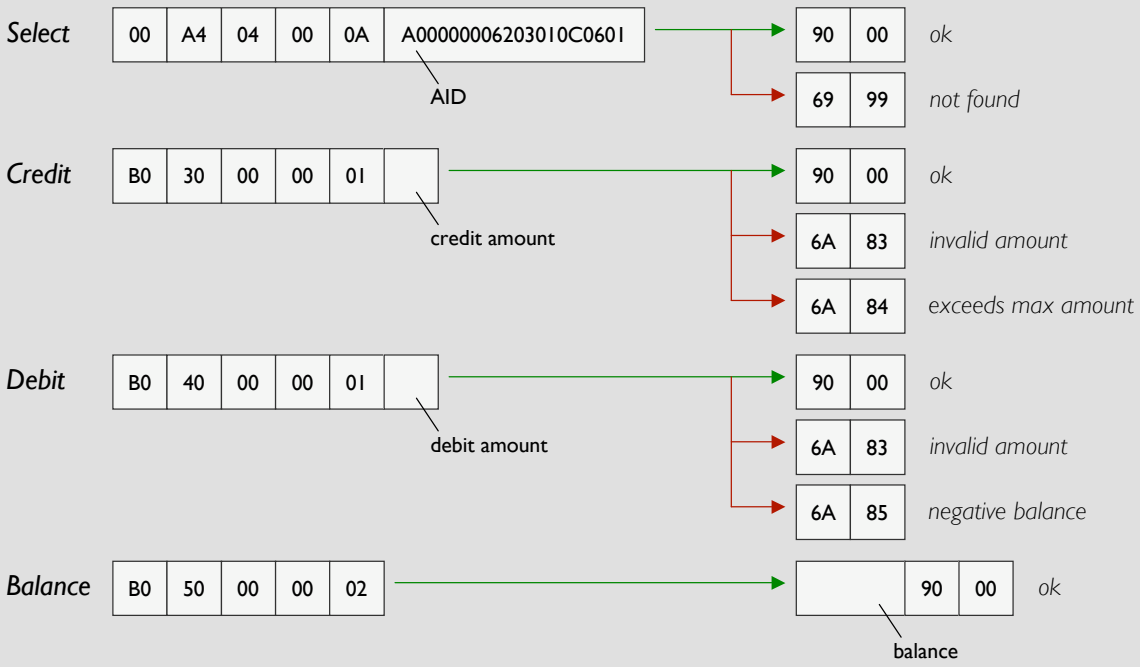
applet design, RMI, size/performance optimizations

D. OO: Applet Design

- Two-step process
  1. *define the command and response APDUs (i.e., the interface between the host application and the card applet)*
  2. *write the applet itself, processing/returning the APDUs specified in 1.*
- Example: Wallet
  - *stores electronic money*
  - *functions: credit, debit, and check balance*

[simplified version of Chen. *Java Card Technology for Smart Cards*. AW, Chapter 12]

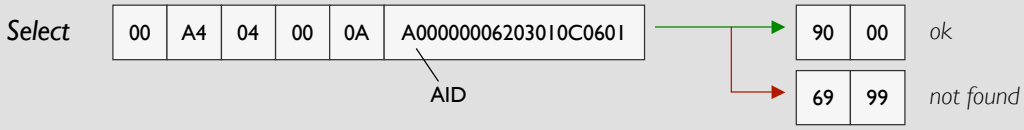
D. OO: Example Wallet



D. OO: Example Wallet

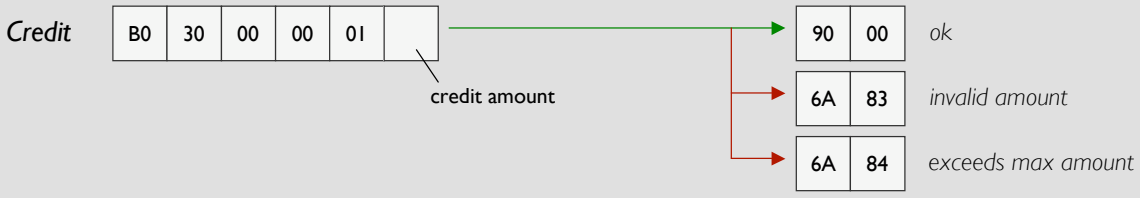
```
public class MyWalletApplet extends Applet {
    private final static byte WALLET_CLA           = (byte)0xB0;
    private final static byte CREDIT_INS           = (byte)0x30;
    private final static byte DEBIT_INS            = (byte)0x40;
    private final static byte BALANCE_INS          = (byte)0x50;

    private final static short SW_INVALID_AMOUNT   = (short)0x6A83;
    private final static short SW_MAX_BALANCE_EXC  = (short)0x6A84;
    private final static short SW_NEG_BALANCE      = (short)0x6A85;

    private final short MAX_BALANCE                = 10000;
    private final short MAX_AMOUNT                 = 100;

    private short balance;

    public static void install(byte[] bArray, short bOffset, byte bLength) {
       (new MyWalletApplet()).register(bArray,(short)(bOffset+1),bArray[bOffset]);
    }

    private MyWalletApplet() {
       // do nothing
    }

    ...
};
```

D. OO: Example Wallet



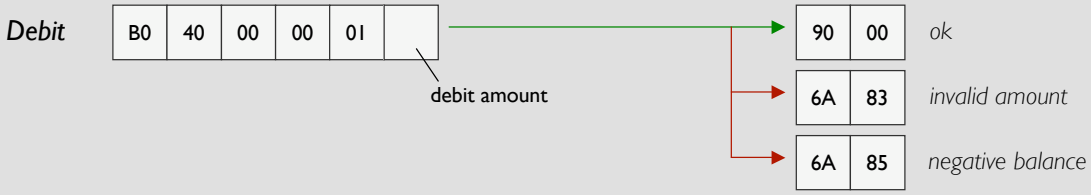

```
public boolean select() {
   return true;
}

public void process(APDU apdu) {
   byte[] buffer = apdu.getBuffer();  // only first five bytes in buffer

   if (selectingApplet())
      return;
   if (buffer[ISO7816.OFFSET_CLA] != WALLET_CLA)
      ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);

   switch (buffer[ISO7816.OFFSET_INS]) {
      case BALANCE: balance(apdu,buffer); break;
      case CREDIT:  credit(apdu,buffer); break;
      case DEBIT:   debit(apdu,buffer); break;
      default: ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
   }
}
```


D. OO: Example Wallet



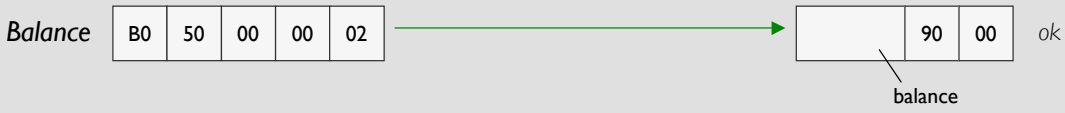

```
public void credit(APDU apdu, byte[] buffer) {
   byte amount;

   // receive data into APDU buffer
   if ( (buffer[ISO7816.OFFSET_LC] != 1) || (apdu.setIncomingAndReceive() != 1) )
      ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
   // check amount
   if ( (amount = buffer[ISO7816.OFFSET_CDATA]) > MAX_AMOUNT) || (amount < 0) )
      ISOException.throwIt(SW_INVALID_AMOUNT);
   // check max balance
   if ((short)(balance+amount) > MAX_BALANCE)
      ISOException.throwIt(SW_MAX_BALANCE_EXC);

   // credit amount
   balance += amount;
}
```

D. OO: Example Wallet




```
public void debit(APDU apdu, byte[] buffer) {
   byte amount;

   // receive data into APDU buffer
   if ( (buffer[ISO7816.OFFSET_LC] != 1) || (apdu.setIncomingAndReceive() != 1) )
      ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
   // check amount
   if ( (amount = buffer[ISO7816.OFFSET_CDATA]) > MAX_AMOUNT) || (amount < 0) )
      ISOException.throwIt(SW_INVALID_AMOUNT);
   // check max balance
   if ((short)(balance-amount) < (short)0)
      ISOException.throwIt(SW_NEG_BALANCE);

   // credit amount
   balance -= amount;
}
```

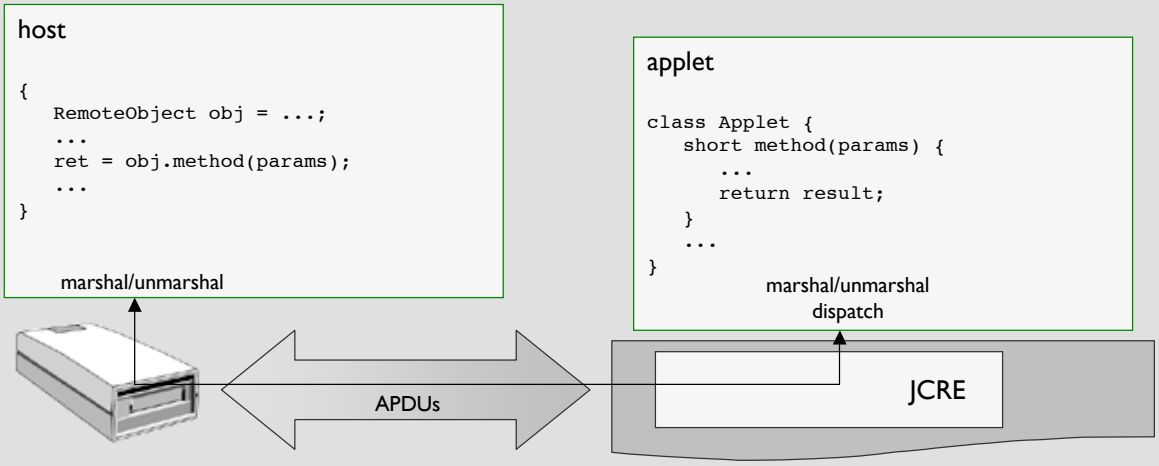

D. OO: Example Wallet




```
public void balance(APDU apdu, byte[] buffer) {
   // inform JCRE that the applet will return some data
   apdu.setOutgoing();

   // set the actual number of outgoing bytes and write balance to APDU buffer
   apdu.setOutgoingLength((short)2);
   Util.setShort(buffer,(short)0,balance);

   // send the two bytes balance
   apdu.sendBytes((short)0,(short)2);
}
```

D. OO: Remote Method Invocation (RMI)

- Communication model based on method calls instead of APDUs
  - *server application makes remote objects accessible [JavaCard]*
  - *client application obtains remote references to these objects and invokes methods on them [host]*



D. OO: JavaCard RMI

- Basic procedure
  - *host requests initial remote object reference for the RMI-based applet*
  - *host sends remote method invocation requests to the card*
  - *RMI messages are encapsulated within APDUs*
- Allowed parameters and return values
  - *parameters*
    - any supported primitive data types
    - any single-dimension array of a supported primitive data type
  - *return values:*
    - any supported primitive data type
    - any single-dimension array type of a supported primitive data type
    - any remote interface type (i.e., remote object reference descriptor)
    - a `void` return

D. OO: JavaCard RMI

- Basic procedure
  - *host requests initial remote object reference for the RMI-based applet*
  - *host sends remote method invocation requests to the card*
  - *RMI messages are encapsulated within APDUs*
- Allowed parameters and return
  - *parameters*
    - any supported primitive data type
    - any single-dimension array of a su
  - *return values:*
    - any supported primitive data type
    - any single-dimension array type o
    - any remote interface type (i.e., re
    - a `void` return

Initial remote object reference retrival

`SELECT` *command returns:*
1. *the byte to be used as instruction byte for subsequent invocations*
2. *the initial remote object reference descriptor (incl. remote object identifier)*

D. OO: JavaCard RMI

- Basic procedure
  - *host requests initial remote object reference for the RMI-based applet*
  - *host sends remote method invocation requests to the card*
  - *RMI messages are encapsulated within APDUs*
- Allowed parameters and return
  - *parameters*
    - any supported primitive data type
    - any single-dimension array of a su
  - *return values:*
    - any supported primitive data type
    - any single-dimension array of a su
    - any remote interface type (i.e., re
    - a `void` return

Method invocation

requires from the host:
1. *the remote object identifier*
2. *the invoked method identifier*
3. *the values of the arguments*

returned to the host:
1. *a primitive return value*
2. *an array of primitive components*
3. *a remote object reference descriptor*
4. *an exception throw by the remote method*

D. OO: JavaCard RMI

- Exceptions thrown by remote methods
  - *for JavaCard-API exceptions, the exception object is transmitted by value (i.e., exact exception type plus embedded reason code)*
  - *for non-JavaCard-API exceptions, the "closest" superclass exception type from the JavaCard API and its embedded reason code are transmitted*
- Limitations
  - *JavaCard RMI is incompatible with Java RMI*
  - *applets cannot invoke remote methods on the host*
  - *method argument data and return values incl. RMI protocol overhead must fit within the APDU size constraints*

D. OO: JavaCard RMI Data Formats

- Remote object reference descriptor
  - *remote object identifier:16 bits unsigned that uniquely identify the remote object on the card*
  - *information to instantiate the proxy class on the host*
- Method identifier
  - *16 bits hash code of a remote method within a remote class*
  - *first 2 bytes of the SHA-1 hash performed on*
    - name of the method
    - class specific hash modifier string
    - method descriptor representation in UTF-8 format
  - *generated by the CAP-file converter (makes sure that there are no collisions)*

D. OO: JavaCard RMI Data Formats

- Remote object reference descriptor
  - *remote object identifier:16 bits u*
    on the card
  - *information to instantiate the pro*
- Method identifier
  - *16 bits hash code of a remote m*
  - *first 2 bytes of the SHA-1 hash f*
    - name of the method
    - class specific hash modifier str
    - method descriptor representa
  - *generated by the CAP-file conver*

Remote object reference descriptor

```
remote_ref_descriptor {
   union {
      ref_null remote_ref_null;
      remote_ref_with_class remote_ref_c
      remote_ref_with_interfaces remote_ref_i
   }
}

ref_null {
   u2 remote_ref_id = 0xffff
}
```


D. OO: JavaCard RMI Data Formats

- Remote object reference descriptor
  - *remote object identifier:16 bits u*
    on the card
  - *information to instantiate the pro*
- Method identifier
  - *16 bits hash code of a remote m*
  - *first 2 bytes of the SHA-1 hash f*
    - name of the method
    - class specific hash modifier str
    - method descriptor representa
  - *generated by the CAP-file conver*

Remote object reference descriptor (cont'd)

```
remote_ref_descriptor {
   union {
      ref_null remote_ref_null;
      remote_ref_with_class remote_ref_c
      remote_ref_with_interfaces remote_ref_i
   }
}

remote_ref_with_class {
   u2 remote_ref_id != 0xffff
   u1 hash_modifier_length;
   u1 hash_modifier[has_modifier_length]
   u1 pkg_name_length
   u1 package_name[pkg_name_length]
   u1 class_name_length
   u1 class_name[class_name_length]
}
```

D. OO: JavaCard RMI Data Formats

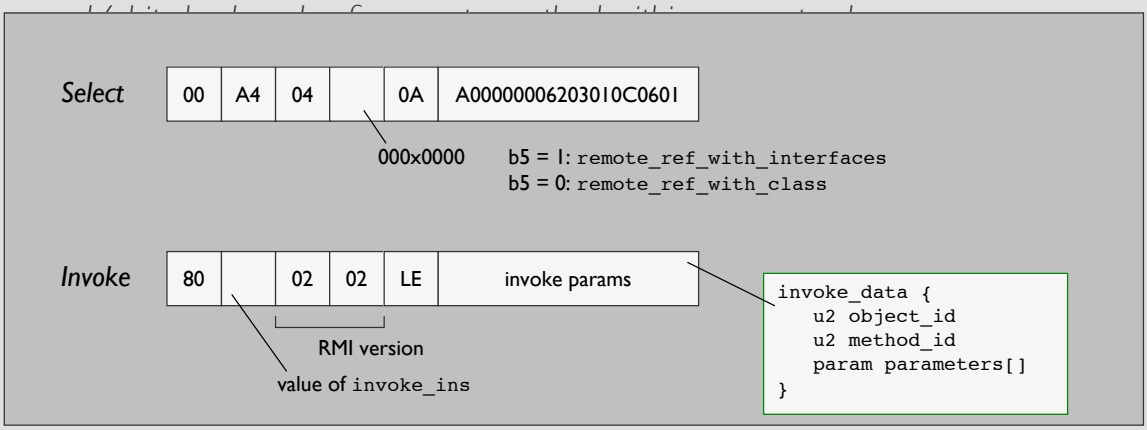
- Remote object reference descriptor
  - *remote object identifier: 16 bits u*
    on the card
  - *information to instantiate the pr*
- Method identifier
  - *16 bits hash code of a remote m*
  - *first 2 bytes of the SHA-1 hash f*
    - name of the method
    - class specific hash modifier str
    - method descriptor representa
  - *generated by the CAP-file conver*

Remote object reference descriptor (cont'd)

```
remote_ref_descriptor {
   union {
      ref_null remote_ref_null;
      remote_ref_with_class remote_ref_c
      remote_ref_with_interfaces remote_ref_i
   }
}

remote_ref_with_interfaces {
   u2 remote_ref_id != 0xffff
   u1 hash_modifier_length
   u1 hash_modifier[hash_modifier_length]
   u1 rem_intf_count
   rem_interface_def rem_intfs[rem_intf_count]
}

rem_interface_def {
   u1 pkg_name_length
   u1 package_name[pkg_name_length]
   u1 interface_name_length
   u1 interface_name[interface_name_length]
}
```


D. OO: JavaCard RMI Data Formats

- Remote object reference descriptor
  - *remote object identifier: 16 bits unsigned that uniquely identify the remote object on the card*
  - *information to instantiate the proxy class on the host*
- Method identifier



D. OO: `javacard.framework, javacard.rmi`

- `javacard.rmi.Remote`
  - *identifies interfaces whose methods may be invoked from an host application*
- `javacard.framework.service.[Basic|RMI]Service`
  - *service framework*
- `javacard.framework.service.CardRemoteObject`
  - *convenient base class for remote objects*

D. OO: `javacard.framework, javacard.rmi`

- `javacard.rmi.Remote`
  - *identifies interfaces whose methods may be invoked from an host application*
- `javacard.framework.`
  - *service framework*
- `javacard.framework.`
  - *convenient base class for*

```
public interface Remote { // tagging interface
}
```

D. OO: javacard.framework, javacard.rmi

- javacard.rmi.Remote
  - *identifies interfaces whose methods may be invoked from an host application*
- javacard.framework.
  - *service framework*
- javacard.framework.
  - *convenient base class for*

```
public interface Service {
    public boolean processCommand(APDU apdu);
    public boolean processDataIn(APDU apdu);
    public boolean processDataOut(APDU apdu);
};

public class BasicService implements Service {
    public boolean fail(APDU apdu, short sw);
    public boolean succeed(APDU apdu);
    public boolean succeedWithStatusWord(APDU apdu)
    public void setStatusWord(APDU apdu, short sw);
    public byte getCLA(APDU apdu);
    public byte getINS(APDU apdu);
    public byte getOutputLength(APDU apdu);
    public byte getP1(APDU apdu);
    public byte getP2(APDU apdu);
    public short getStatusWord(APDU apdu);
    public short receiveInData(APDU apdu);
    public boolean selecingApplet(APDU apdu);
    public void setOutputLength(APDU apdu, short length);
    public boolean isProcessed(APDU apdu);
    public void setProcessed(APDU apdu);
}
```


D. OO: javacard.framework, javacard.rmi

- javacard.rmi.Remote
  - *identifies interfaces whose methods may be invoked from an host application*
- javacard.framework.
  - *service framework*
- javacard.framework.
  - *convenient base class for*

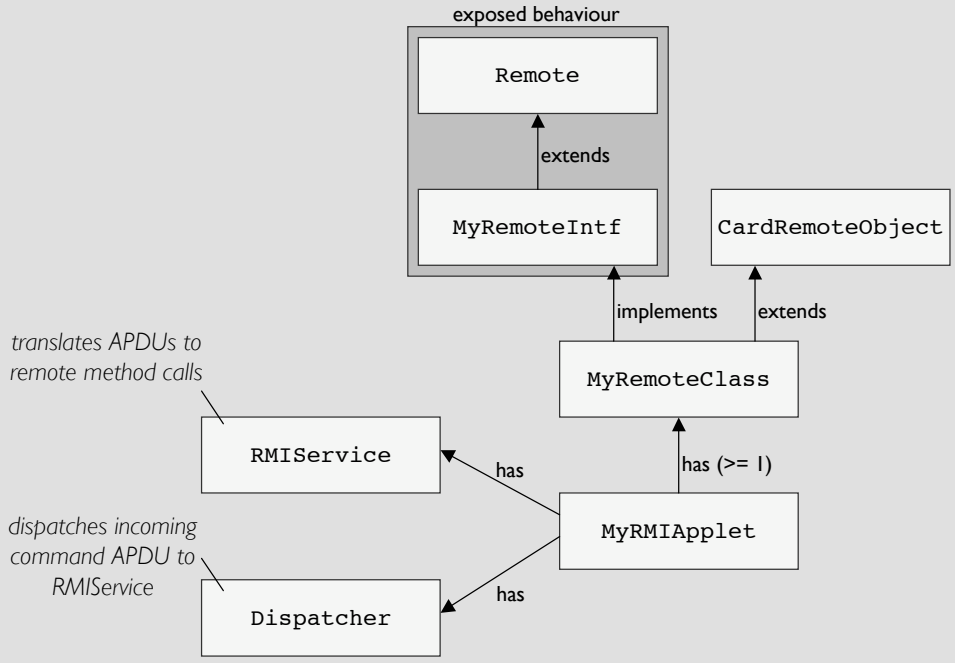
```
public interface RemoteService extends Service;

public class RMIService extends BasicService
                        implements RemoteService {
    public void processCommand(APDU apdu);
    public void setInvokeInstructionByte(byte ins);
};
```

D. OO: `javacard.framework, javacard.rmi`

- `javacard.rmi.Remote`
  - *identifies interfaces whose methods may be invoked from an host application*
- `javacard.framework.`
  - *service framework*
- `javacard.framework.`
  - *convenient base class for*

```
public final class CardRemoteObject implements Remote {
    public static void export(Remote obj);
    public static void unexport(Remote obj);
};
```


D. OO: RMI Applet Structure



D. OO: Example MyRMIApplet

```
public interface MyRemoteInterface extends Remote {
    public short getBalance() throws RemoteException;
    ...
};

public class MyRemoteClass extends CardRemoteObject implements MyRemoteInterface {
    private short balance;

    public short getBalance() throws RemoteException {
      return balance;
    }
    ...
};
```


D. OO: Example MyRMIApplet

```
public class MyRMIApplet extends Applet {
    private Dispatcher disp;
    private RemoteService rsvc;

    public MyRMIApplet() {
      disp = new Dispatcher((short)1);
      disp.addService(rsvc = new RMIService(new MyRemoteClass()),PROCESS_COMMAND);
    }

    public static void install(byte[] bArray, short bOffset, byte bLength) {
      (new MyRMIApplet()).register(bArray,(short)(bOffset+1),bArray[bOffset]);
    }

    public void process(APDU apdu) {
      disp.process(apdu);
    }
};
```

DEMO

D. OO: Optimizations

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

DEMO

Minimize persistent writes

- *Writing to RAM is about 1000 times faster than writing to EEPROM;*
- *Writing to EEPROM is transactional (most of the time);*
- *RAM is not subject to wear.*

Store intermediate results or frequently updated temporary data in RAM!

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

Classes, interfaces, inheritance

PRO
- *Modularity and reusability;*
- *Superclasses encapsulate generic and/or common behaviour, subclasses add specialized behaviour;*
- *Who does not want to program object-oriented, right?!*

CONS
- *A smart card is not your desktop machine (CPU, memory);*
- *Each class/interface adds some memory overhead;*
- *Inheritance and virtual method calls add memory and run-time overhead;*
- *Typical applets are smaller than 16KB, this is not where OO shines!*

Balance your design choices!

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider `switch` vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

Re-use objects

In Java
- *Object are created as needed and garbage collection if no longer used;*
- *All objects are destroyed when the VM terminates.*

In JavaCard
- *Garbage collection is an optional feature in the first place;*
- *All objects exist during the lifetime of the card (applet);*
- *Applets shall never instantiate objects with the expection that their storage will be reclaimed.*

Reuse object instantiations by writing new values to their member variables!

Most likely does NOT work:

```
public void myMethod {
   Object a = new Object();
   ...
}
```

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

Re-use objects: Example `javacard.framework.ISOException`

```
public class ISOException extends CardRuntimeException {
   private static ISOException systemInstance;

   ISOException(short sw) {
      super(sw);
      systemInstance = this;
   }

   public static void throwIt(short sw) {
      systemInstance.setReason(sw);
      throw systemInstance;
   }
}
```


D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider **`switch`** vs. **`if-else`**
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of r
- Consider `switch` vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

Allocate during applet construction

Whenever possible, an applet should create in its constructor all objects it might need during its lifetime!

Advantages:

- *No out-of-memory later;*
- *If the installation fails, the applet's memory is reclaimed (the* `install` *method is transactional).*

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider `switch` vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

Method parameters and local variables

– *The stack holds*
  method parameters,
  return values,
  local variables, and
  partial results;
– *The call stack is only about 200 bytes on most smart cards;*
– *Maximal nesting of method calls is less than 20 on most smart cards.*

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider `switch` vs. `if-else`
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

DEMO

`switch` vs. `if-else`

Sometimes **`switch`** *and* **`if-else`** *can be used interchangeably.*

- *in general:* **`switch`** *executes faster and requires less memory*
- *but sometimes:* **`switch`** *might take more memory*

Try both and use empirical values!

Sometimes its even best to mix both:

```
if (ins == VERIFY_PIN)
   verifyPin(apdu);
else {
   if (!isPinValidated))
     ISOException.throwIt(SW_PIN_NOT_VALIDATED);

   switch (ins) {
      ...
   }
}
```


D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider **`switch`** vs. **`if-else`**
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

Initialization of statics and instance variables

Basically, initialization is enforced by the Java compiler!

- *Local variables must be explicitly initialized;*
- *Statics and instance variables have default initializers.*
  - *object references:* `null`
  - *boolean:* `false`
  - *primitive types:* `0`

CAP-file converter optimizations:

- `static` *field initializations are dumped as binary image*
  e.g., `static byte b[] = { 1, 2, 3 };`
- `final static` *fields are inlined*
  loading a `byte` *constant is cheaper than referencing a* `static` *field*
  loading a `short` *constant has no impact on code size.*

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider **`switch`** vs. **`if-else`**
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

DEMO

Compound statements

Re-use temporary values and return values wherever possible.

```
x = a+b;
x = x-c;
```

vs.

```
x = a+b-c;
```

Note: Nesting increases the operand stack size and may degrade readability.

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider **`switch`** vs. **`if-else`**
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

DEMO

Local variables as value cache

Accessing an array element or an object field requires more byte codes than accessing a local variable.

```
if (buffer[ISO7816.OFFSET_INS] == VERIFY)
   verifyPin(apdu);
else if (buffer[ISO7816.OFFSET_INS] == CREDIT)
   credit(apdu)
else if (buffer[ISO7816.OFFSET_INS] == DEBIT)
   ...
```

vs.

```
byte ins;
if ((ins = buffer[ISO7816.OFFSET_INS]) == VERIFY)
   verifyPin(apdu);
else if (ins == CREDIT)
   credit(apdu)
else if (ins == DEBIT)
   ...
```


D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces, and inheritance wisely
- Re-use objects instead of allocating new ones
- Allocate your objects in the applet's constructor
- Optimize the use of method parameters and local variables
- Consider **`switch`** vs. **`if-else`**
- Don't initialize statics and instance vars with their default values
- It's a stack machine, so work on the stack
- Use local variables to cache the values of arrays or object fields
- Operate on the first 4 local variables whenever possible

D. OO: Optimizations

- Minimize writing to persistent memory
- Use classes, interfaces
- Re-use objects instea
- Allocate your objects
- Optimize the use of
- Consider **`switch`** vs.
- Don't initialize statics
- It's a stack machine, s
- Use local variables to
- Operate on the first

DEMO

Focus on the first 4 local variables

For reading and writing the first 4 local variables of each method, specific byte codes are reserved.

```
aload_<n>
astore_<n>
sload_<n>
sstore_<n>

iload_<n>
istore_<n>
```