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

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Memory Model

- Three kinds of memory
  - *Transient memory (RAM)*
    - read/write memory
    - looses contents in case of power loss
    - most expensive memory type
  - *Persisent memory (EEPROM or Flash)*
    - read/write memory
    - preserves contents in case of power loss
    - number of write operations is physically limited
    - write/erase is significantly slower than in RAM
  - *"System memory" (ROM or Flash)*
    - read-only memory
    - preserves contents in case of power loss
    - cheapest memory type

RAM
up to 8 KB

EEPROM, Flash
up to 96 KB | 1 MB

ROM
up to 256 KB

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are preserved across sessions*
- Transient objects
  - *memory is preserved across sessions but data is erased between sessions*
- JCRE (Temporary) Entry-Point Objects
  - *objects owned by the JavaCard system which may be called from any context*
  - *references to temporary entry-point objects (e.g.,* `APDU`*) may not be stored*

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are prese*
- Transient objects
  - *memory is preserved across*
- JCRE (Temporary) Entry-
  - *objects owned by the JavaCa*
  - *references to temporary entr*

Persistent and transient objects in Java:

- *in Java, all objects are created in RAM*
- *object (de)serialization records/restores the state and properties of an object in a stream of bytes*
- *the* `transient` *keyword indicates which fields are not part of an object's persistent state*

JavaCard does support neither (de)serialization nor the `transient` *keyword.*

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are prese*
- Transient objects
  - *memory is preserved across*
- JCRE (Temporary) Entry-
  - *objects owned by the JavaC*
  - *references to temporary entr*

Persistent JavaCard objects:

- *created by the* `new` *operator*
- *updates to single fields are atomic*
- *can be referenced by fields in a transient objects*
- *fields can reference transient objects*
- *if not referenced by some other object it becomes unreachable and may be garbage collected*

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are prese*
- Transient objects
  - *memory is preserved across*
- JCRE (Temporary) Entry-
  - *objects owned by the JavaC*
  - *references to temporary entr*

Transient JavaCard objects:

- *created by invoking the JavaCard APIs*
- *only arrays with primitive types or with references to* `Object`
- *updates to single fields are not atomic*
- *can be referenced by fields in persistent objects*
- *fields can reference persistent objects*
- *if not referenced by some other object it becomes unreachable and may be garbage collected*

- *two types:* `CLEAR_ON_RESET`, `CLEAR_ON_DESELECT`

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are prese*
- Transient objects

`CLEAR_ON_RESET`

- *used for maintaining data to be preserved across applet selections but not across card resets*
- *on card reset all fields are cleared*

Transient JavaCard objects:

- *created by invoking the JavaCard APIs*
- *only arrays with primitive types or with references to* `Object`
- *updates to single fields are not atomic*

can be referenced by fields in persistent objects
fields can reference persistent objects
if not referenced by some other object it becomes unreachable and may be garbage collected

two types: `CLEAR_ON_RESET`, `CLEAR_ON_DESELECT`

Copyright © 2004-2007 IBM Corp.

B. Memory: JavaCard Object Model

- Java programming rules for objects
  - *all objects are instances of classes or array types w/ root* `java.lang.Object`
  - *fields in a new object or components in a new array are initialized to their default values* `0`, `null`, `false` *unless explicitly specified*
- Persistent objects
  - *memory and data are prese*
- Transient objects

`CLEAR_ON_DESELECT`

- *used for maintaining data NOT to be preserved across applet selections or card resets*
- *on applet deselect all fields are cleared*

`CLEAR_ON_DESELECT` *implies* `CLEAR_ON_RESET`

Transient JavaCard objects:

- *created by invoking the JavaCard APIs*
- *only arrays with primitive types or with references to* `Object`
- *updates to single fields are not atomic*

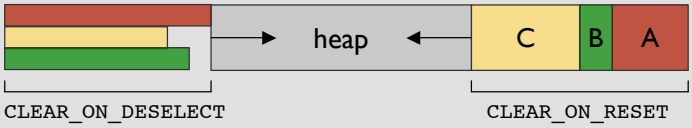
can be referenced by fields in persistent objects
fields can reference persistent objects
if not referenced by some other object it becomes unreachable and may be garbage collected

two types: `CLEAR_ON_RESET`, `CLEAR_ON_DESELECT`

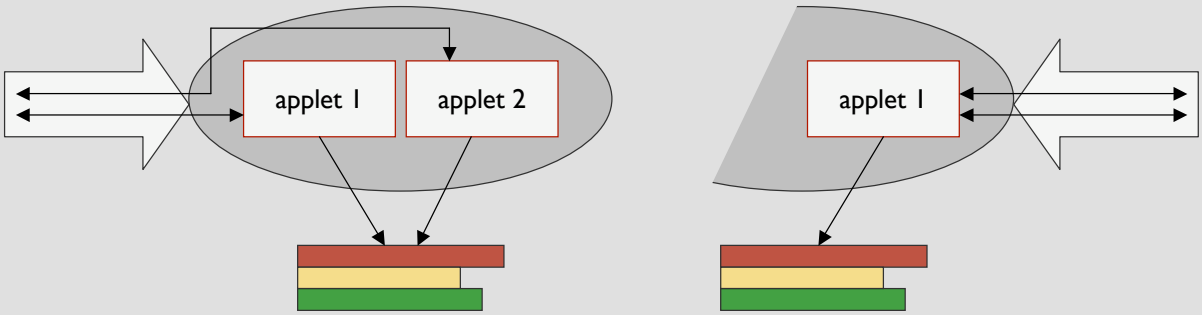
Copyright © 2004-2007 IBM Corp.

B. Memory: Organization

- Memory layout: `CLEAR_ON_RESET` vs. `CLEAR_ON_DESELECT`



- Multi-selectable
  - *multi-selected applets from the same package share their COD*



Copyright © 2004-2007 IBM Corp.

B. Memory: Garbage Collection

- Traditional: `malloc` and `free`
- Reference counting
  - *each object is associated with a count of how many objects currently use it*
  - *counter is increased and decreased by* `retain` *and* `release`*, respectively*
  - *if counter reaches 0, the object is deleted*
- Garbage collection
  - *system automatically keeps track of who uses which objects*
  - *periodically reclaims memory of unreferenced objects*
  - *JavaCard*
    - introduced as optional feature in JavaCard 2.2
    - invoked via the JavaCard API
    - may be delayed until the next invocation of `Applet.process()`

Copyright © 2004-2007 IBM Corp.

B. Memory: `javacard.framework`

- `javacard.framework.JCSystem`
  - *collection of methods to control applet execution, memory management, atomic transaction management [2.C], inter-applet object sharing [3.A]*

Copyright © 2004-2007 IBM Corp.

B. Memory: `javacard.framework`

- `javacard.framework.JCSystem`
  - *collection of methods to control applet execution, memory management, atomic transaction manag*

```
public final class JCSystem {
    public static boolean[] makeTransientBooleanArray
          (short length, byte event);
    public static byte[] makeTransientByteArray
          (short length, byte event);
    public static short[] makeTransientShortArray
          (short length, byte event);
    public static Object[] makeTransientObjectArray
          (short length, byte event);

    public static byte isTransient(Object theObject);
    public static short getAvailableMemory(byte type);

    public static short getVersion();
    public static AID getAID();

    public static boolean isObjectDeletionSupported();
    public static void requestObjectDeletion();

    ...
};
```


Copyright © 2004-2007 IBM Corp.