INFO213: Lecture 4

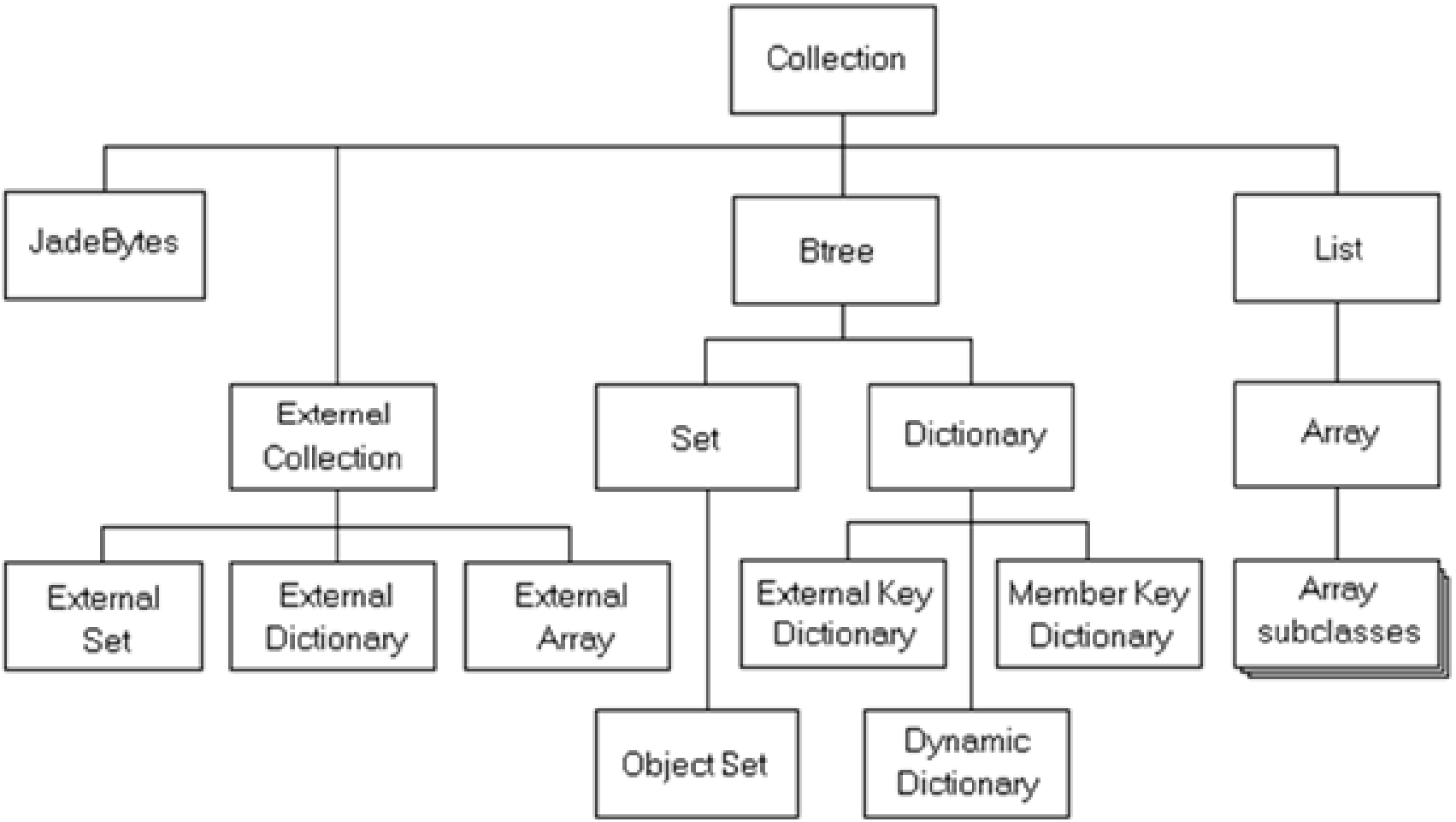
JADE Collections, Testing, and (more) GUI

Plus, some Assignment tips

JADE Collections

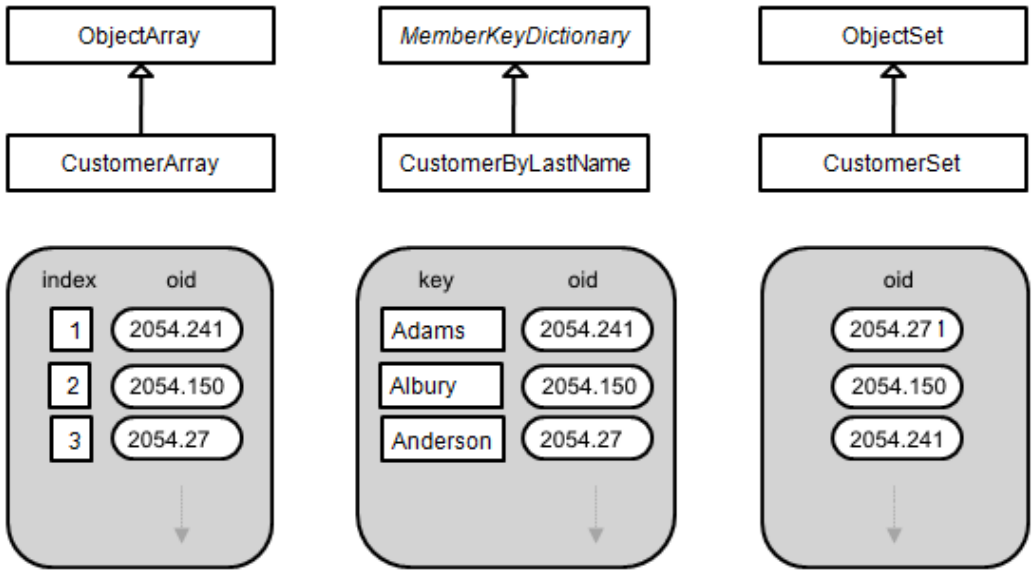
- What constitutes a relational database?
- What constitutes an object database?
  - Collections (of … things, objects)
    - One-to-many relationships
    - Many to many relationships

JADE Collection Hierarchy



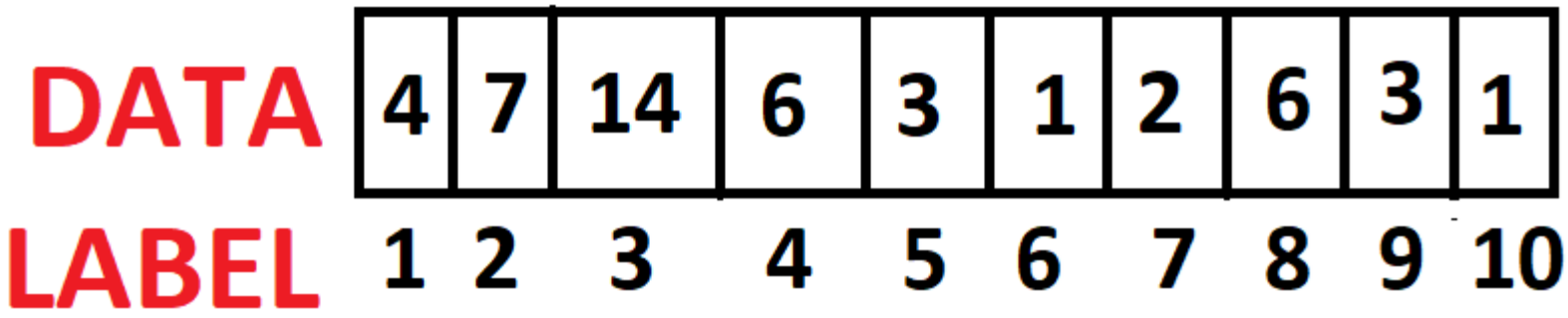
Collection Types Definitions

- **Array:** a collection of objects or primitive values, ordered by index number
- **Dictionary:** a collection of objects ordered by keys
  - **MemberKeyDictionary:** keys are properties of the member objects
  - **ExtKeyDictionary:** keys are specified manually when objects are added
  - **DynaDictionary:** a dictionary defined at run time
- **Set:** a collection of objects conceptually unordered (in practice, ordered by OID)



Array

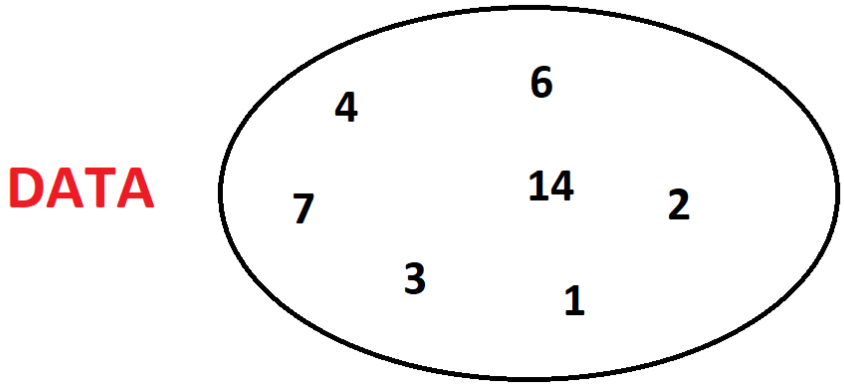
- **Data:** Can be any type - String, Integer, Date, or even any Class
- **Label:** Do arrays begin at 0?



- Array subclass determines its type (StringArray, IntegerArray CustomerArray etc.)

Set

- **Data only, no labels**
- **No duplicates allowed!**
- **Something is either "in the set" or not.**



- Sets are **"conceptually unordered"** - realistically they will accidently happen into an order (oid) during implementation, but think of them as unordered.

Dictionary

- **Each data point has a key and a value**
- **Searchable by key**
- **Keys must be unique.**
- **Values don't have to be...**

Key	Value
Aaron	4
Betty	14
Carl	1
Daisy	4
Earnest	7
Faye	14
Gerry	3

- Dictionaries are good for easily looking up data.
- Consider a classic dictionary - has words and for each word, a definition

Common Collections Methods

- **add(obj)** – Adds an object to a collection
- **clear()** – Removes all entries from a collection
- **first()** – Returns the first entry in the collection
- **includes(obj)** – Returns true if the collection contains a specified object
- **last()** – Returns the last entry in the collection
- **purge()** – Deletes all objects referenced in a collection
- **remove(obj)** – Removes an item from a collection
- **size()** – Returns the current number of entries in the collection
- For a complete list of methods search for “collection methods” in JADE Help

foreach: Iterating Over Collections

- Using *foreach* instruction: syntax

```
foreach variable in collection-expression [options]
    [where condition] do [:label]
    [foreach-instructions]
endforeach [label];
```

- Example*:*

```
foreach customer in customerDict do
    write customer.getFullName;
endforeach;
```

- *continue* instruction
  - Skip the rest of the loop body and start next iteration
- *break* instruction
  - Terminate the loop and proceed with code execution

Iterating Over Collections Smarter

- Using *while* and collection *Iterator: syntax*

  while *condition* do *[:label]*
  [while-instructions]
  endwhile *[label];*

- Example:

```
cust: Customer; // Declared in the vars section.
iter: Iterator; // Declared in the vars section.
iter := customerDict.createIterator();
while iter.next(cust) do
    write customer.getFullName();
endwhile;
```

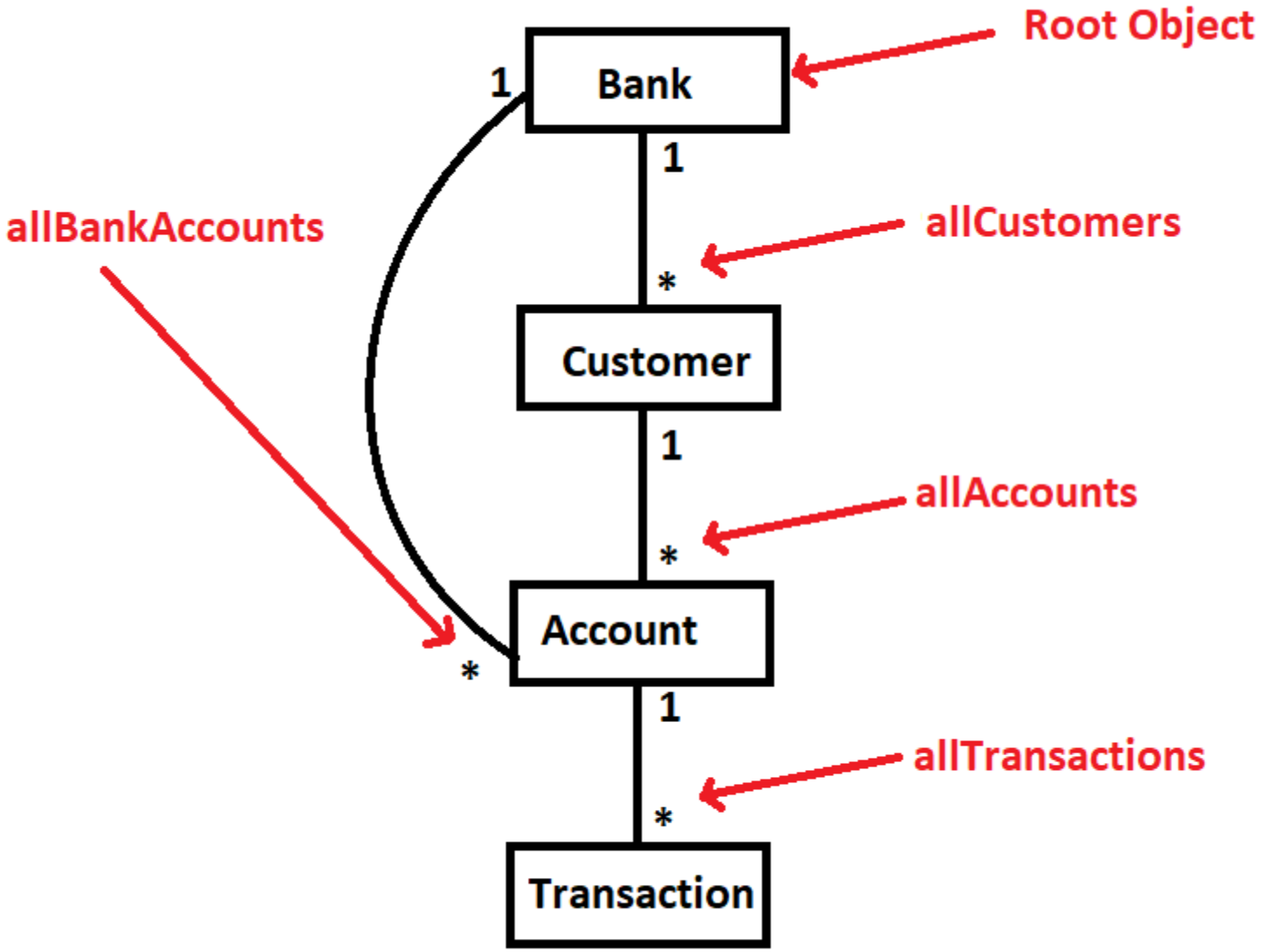
- Iterator methods:
  - startAtIndex, startAtObject (among others)

Iterators vs. *foreach*

- By default, *foreach* share-locks the collection
  - Other processes cannot modify the collection
- Which of the two is better? – It depends…
  - Performance is equivalent in basic cases
  - *Iterator* is suitable where locking does not matter
  - Filtering: *foreach* allows selective attribute value restrictions
  - *Iterator* allows arbitrary starting positions
- Verdict: the choice between the two is determined by the context

Building Up the Collections/Database

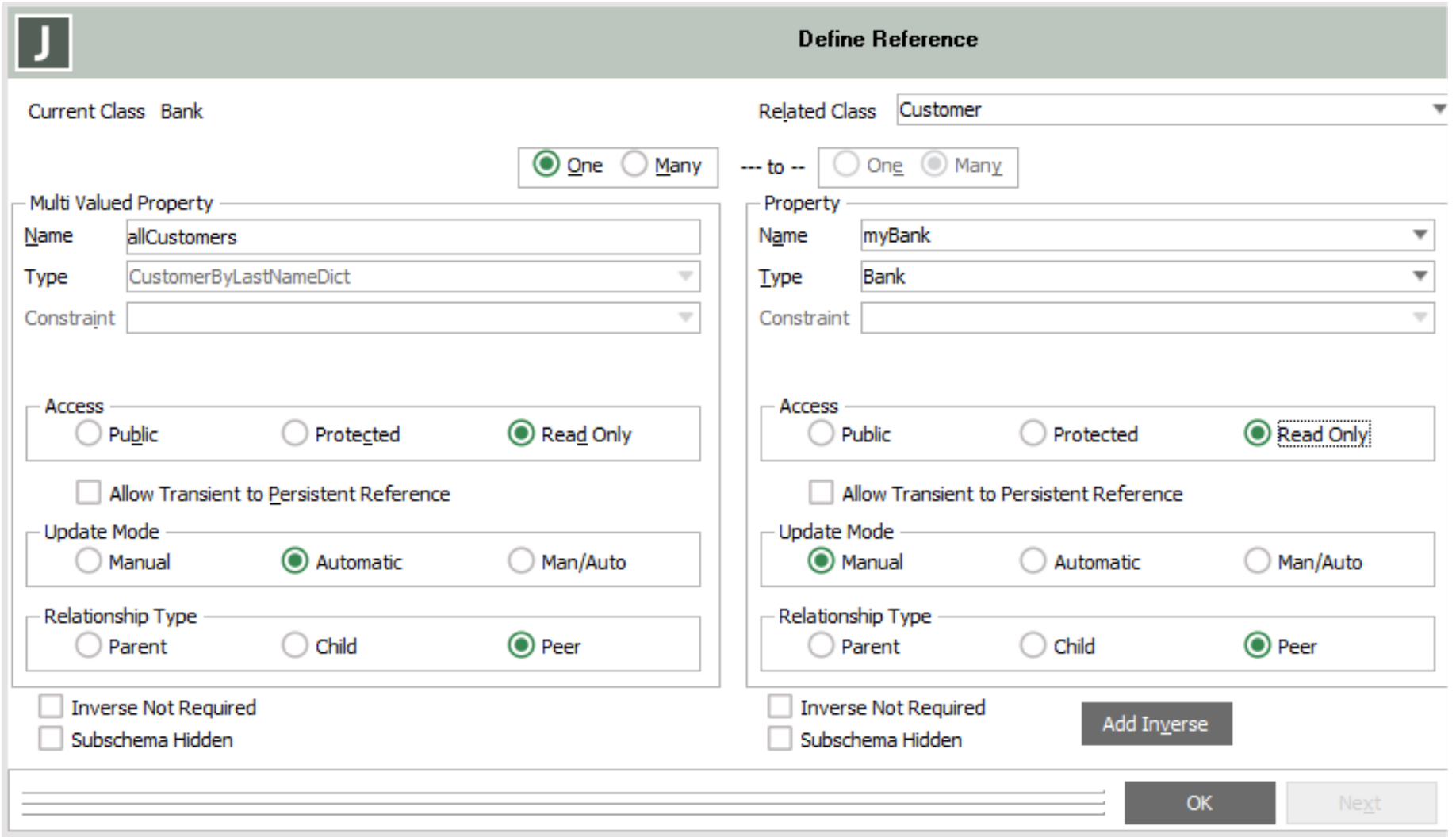
- One-to-many references
- Comprehensive collections in the root object (Bank)
  - *allCustomers*
  - New customers inserted in *Customer* constructor code: *self.myBank.allCustomers.add(self);* *?? Can we do better?*
  - *allBankAccounts* (in the near future)
- *Account* collection in the Customer class
  - allAccounts (in the near future)
- *Transaction* collection in the *Account* class
  - *allTransactions (for future development)*


Root Object
Bank
1
1
allBankAccounts
allCustomers
*
Customer
1
allAccounts
*
Account
*
1
allTransactions
*
Transaction

Smarter Collections Definitions

- Exclusive collections option
  - Owned by the parent object
  - Created and deleted with the parent object
- Defining inverses
  - Automatic vs. manual updating
  - Automatic updating is essential for data integrity
  - Automatic updating reduces the amount of code
    - In Customer constructor:
      self.myBank := app.myBank; // Our code.
      self.myBank.allCustomers.add(self); // Automatic maintenance.

Automatic/Manual Updating Settings



Virtual Collections

- Read-only property of the *Class* class (yes!) keeps track of all instances of this class

```
purgeTestObjects() updating;

begin
    beginTransaction;

    // Remove all instances of CustomerByLastNameDictionary class.
    CustomerByLastNameDict.instances.purge();

    // Remove all instances of Customer class.
    Customer.instances.purge();

    // Remove the single instance of the Bank class.
    delete Bank.firstInstance;

    commitTransaction;
end;

```

instances Property Use in Code

- Use of the instances property with normal collection methods.
- *// Count instances.*
  count := Employee.instances.size;
- *// Delete all instances.*
  Employee.instances.purge();
- *// First instance.*
  emp := Employee.instances.first.Employee;
- *// Last instance.*
  cust := Customer.instances.last.Customer;
- *// Copy to another collection.*
  Product.instances.copy(productCollection);

Testing

- Test early, test often
- Your code isn't "done" unless it has unit tests
- Links back to analysis/design/development process:
  - Analysis gives *requirements*
  - Development gives *code*
  - Tests PROVE that your *code* fulfils the *requirements* (like a contract)
  - You can even write them BEFORE the code! (test-driven development)

Testing

- Test engineer is a common starting position
  - Most developers start as testers
  - **All** developers are expected to write tests as part of their job
- In some cases tests run continuously on new code
  - Developers can run test suites to find what their code breaks
- JADE provides a very handy testing framework

Manual vs Automated

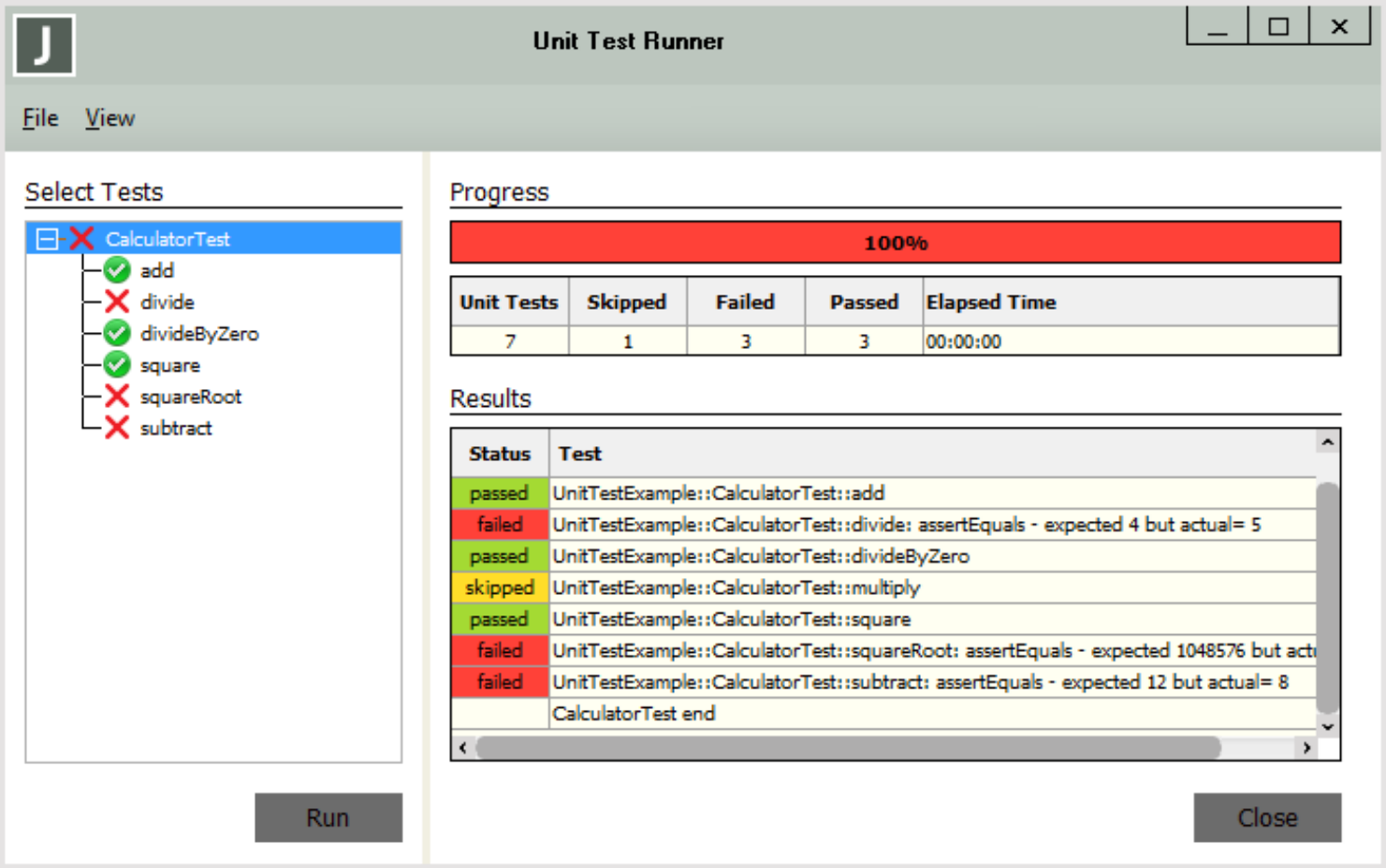
- A Manual test is one that you have to perform yourself every time
  - You better be pretty sure you're not gonna have to repeat manual tests many times!
- An automated test runs, well, automatically!
  - Can be scripted to run every night
  - Can just press a single button and run some or all of the tests
  - More effort the first time, but repeatable.

End-to-end vs Unit tests

- End-to-end tests test the whole application
  - e.g. clicking through the forms to perform actions
  - Usually done manually, can be automated by ATCG
- Unit tests test one very small part (unit) of the code (usually a single method)
  - Usually automated
  - Can be done in the Jade Test Framework
  - Will need to do this in the lab test

Unit Test Runner: Demo

- Result of all tests in one view



GUI Demo – ListBox and Table

displayCollection

Signature

```
displayCollection(c:          Collection;     (Table)
                  update:     Boolean;
                  showHow:    Integer;
                  startObj:   Object);

displayCollection(c:          Collection;     (ComboBox, ListBox)
                  update:     Boolean;
                  showHow:    Integer;
                  startObj:   Object;
                  extraEntry: String);
```