Instructor: Prasun Dewan
PREREQUISITE

- Arrays Collections Implementation
COLLECTION TYPES

- StringHistory, StringDatabase, StringSet
- Array
- ArrayList, List
- Map
- Stack
- Queue
**STRUCTURED VS. ATOMIC TYPES**

**Atomic Types**
- Primitive types:
  - `double`
  - `int`
- Classes:
  - ABMICalculator
- Interfaces:
  - BMICalculator

**Structured Types**
- Classes:
  - ACartesianPoint
  - StringHistory
- Interfaces:
  - Point
  - AStringHistory

Instances of structure type decomposed into one or more smaller values
Logical Structure Point

- X: int
- Y: int
- Angle: double
- Radius: double

Property name:

Interface, Class or primitive type of property value
Logical Structure Array

{5, 6, 7, 8}

0 1 2 3

int int int int

index

Interface, Class or primitive type of property value
Components can be of different types and serve different functions.

Components are of same type (which may be a super type of the specific types of the components) and are handles in the same way.

Object[] objects = { “Joe Doe”, new AStringDatabase(), new AStringHistory() }
**Bean vs. (Indexed) Collection**

- **Bean**: Program fragment can refer to only a particular kind of component of composite object.
- **Collection**: Program fragment can refer to any component of the composite object.
- **Indexed collection**: Components referred explicitly by int expressions.

Program fragment:
- `point.getX()`
- `point.setY(100)`
- `scores[position + 1]`
INDEXED COLLECTIONS

Each element identified by a unique index

Successive elements have consecutive indices
STRING HISTORY?

size()
elementAt()
addElement()
StringHistory

stringHistory

Collection: Program fragment can refer to any component of the composite object

Indexed collection: components referred explicitly by int expressions

stringHistory.elementAt(position + 1)
**String History vs. Array**

Dynamic: After creation, can grow

Programmer-defined

```
String[] strings = { "James Dean", "Joe Doe", "Jane Smith" };```

Static and Language-defined

```
<terminated> AHistoryDriver [Java Application]
James Dean
p
******************************
James Dean
******************************
Joe Doe
Jane Smith
******************************
James Dean
Joe Doe
Jane Smith
******************************
p
```
STATIC VS. DYNAMIC STRUCTURES

**Static**

- Beans have fixed number of properties
- Arrays have fixed number of elements
- Though an array variable can be assigned arrays of different sizes

**Dynamic**

- Can create new edges in logical structure
public interface StringHistory {
    public void addElement(String element);
    public int size();
    public String elementAt(int index);
}
**DATABASE**

```java
public interface StringDatabase {
    // from history
    public int size();
    public void addElement(String element);
    public String elementAt(int index);

    // additional methods
    public void removeElement(String element);
    public boolean member(String element);
    public void clear();
}
```

**Do we need a history if we have a database?**

Yes, principle of least privilege
PRINCIPLE OF LEAST PRIVILEGE/ NEED TO KNOW

- Do not give a user of some code more rights than it needs
  - Code is easier to change
  - Need to learn less to use code
  - Less likelihood of accidental or malicious damage to program
- Like hiding engine details from car driver
Visibility of a Type and its Members: Some Type are More Equal than Others

Subtypes Co-Packaged

All classes

Increasing access

private
default
protected
public

- Co-packaged a la co-workers, amplifier + speakers
- Subtype a la family, deluxe amplifier

Encapsulation Rule: Do not make variables of a class public
Using Database as History

public interface StringDatabase {
    // from history
    public int size();
    public void addElement(String element);
    public String elementAt(int index);

    // additional methods
    public void removeElement(String element);
    public boolean member(String element);
    public void clear();
}

Programmer would be able to perform inappropriate operations on a logical history implemented physically as a database.
public interface StringDatabase extends StringHistory {
    //additional methods
    public void removeElement(String element);
    public boolean member(String element);
    public void clear();
}

Programmer would be able to perform inappropriate operations on a logical history implemented physically as a database
**Vector: General Object Collection**

```java
public final int size();
public final Object elementAt(int index);
public final void addElement(Object obj);
public final void setElementAt(Object obj, int index);
public final void insertElementAt(Object obj, int index);
public final boolean removeElement(Object obj);
public final void removeElementAt(int index);
public final int indexOf(Object obj);
...
```

---

**Do we need other collections if we have Vector**

- Yes, principle of least privilege
- Yes, implementation considerations
Class ArrayList and Vector (List)

```java
public final int size();
public final Object get(int index);
public final void add(Object obj);
public final void set(int index, Object obj);
public final void insert(int index, Object obj);
public final boolean remove(Object obj);
public final void remove(int index);
public final int indexOf(Object obj);
...
```

Vector has ArrayList (List) methods plus the additional original methods in the previous slides

Can add arbitrary objects to these collections
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
public class VectorArrayListUser {
    public static void main (String[] args) {
        List names = new Vector();
        List grandSlams = new ArrayList();
        names.add("Nadal");
        grandSlams.add(13);
        names.add("Federer");
        grandSlams.add(17);
        names.add("Borg");
        grandSlams.add(11);
        names.add("Sampras");
        grandSlams.add(14);
    }
}
Indexed Collections

Each element identified by a unique index

Successive elements have consecutive indices
Tables

Each element identified by a unique object called a key

Usually strings are used as keys
// associates key with value, returning last value associated with key
public final Object put (Object key, Object value);
// returns last value associated with key, or null if no association
public final Object get (Object key);

Final means method cannot be overridden
```java
public static void main (String[] args) {
    Map aMap = new HashMap();
    aMap.put("Nadal", 10);
    aMap.put("Federer", 17);
    aMap.put("Sampras", 14);
    System.out.println(aMap.get("Nadal"));
    System.out.println(aMap.get("nadal"));
    aMap.put("Nadal", 11);
    System.out.println(aMap.get("Nadal"));
    System.out.println(aMap);
}
```

Output:

```
10
null
11
{Sampras=14, Federer=17, Nadal=11}
```
**Explicit vs. Implicit Element Reference**

- **Collection**: Program fragment can refer to any component of the composite object.

- **Indexed collection**: Components referred explicitly by int expressions.

- **Table collection**: Components referred explicitly by Object expressions.

- **Implicit reference to components?**
STACK: LAST IN FIRST OUT

public interface StringStack {
    public boolean isEmpty();
    public String getTop();
    public void push(String element);
    public void pop();
}

StringStack stringStack = new AStringStack();
stringStack.push("James Dean");
stringStack.push("Joe Doe");
stringStack.push("Jane Smith");
stringStack.push("John Smith");
System.out.println(stringStack.getTop());
stringStack.pop();
System.out.println(stringStack.getTop());

John Smith
Jane Smith
Queue: First in First Out

```java
public interface StringQueue {
    public boolean isEmpty();
    public String getHead();
    public void enqueue(String element);
    public void dequeue();
}

StringQueue stringQ = new AStringQueue();
stringQ.enqueue("James Dean");
stringQ.enqueue("Joe Doe");
stringQ.enqueue("Jane Smith");
stringQ.enqueue("John Smith");
System.out.println(stringStack.getHead());
stringQ.dequeue();
System.out.println(stringStack.getHead());
```

James Dean
Joe Doe
STRUCTURED TYPES

Static named

Static indexed

Dynamic indexed

(Static) Dynamic tables

Stack (LIFO)

Queue (FIFO)
**Read-only and Editable Properties**

Typed, Named Unit of Exported Class State

```java
public class C {
    public static T getP() {
        ...
    }

    public static void setP(T newValue) {
        ...
    }
}
```

**Typed, Named Unit of Exported Class State**

**public class C**

**Name P**

**Type T**

**Read-only**

**Editable**

**Getter method**

**Setter method**

**Bean convention:** For humans and tools

**Violates Bean convention**
public interface StringHistory {
    public void addElement(String element);
    public int size();
    public String elementAt(int index);
}

public static void main (String[] args) {
    StringHistory stringHistory = new AStringHistory();
    stringHistory.addElement("James Dean");
    stringHistory.addElement("Joe Doe");
    stringHistory.addElement("Jane Smith");
    stringHistory.addElement("John Smith");
    ObjectEditor.edit(stringHistory);
}
CONVENTIONS FOR VARIABLE-SIZED COLLECTION

@StructurePattern(StructurePatternNames.VECTOR_PATTERN)
public interface C{
    public T elementAt (int index);
    public int size();
    public Any setElementAt(T t, int index);
    ...
}

Convention based on Vector

Write method (optional)

Arbitrary Type.

Read methods

Unconstrained Type (void or T in practice)
ALTERNATIVE CONVENTIONS FOR VARIABLE-SIZED COLLECTION

@StructurePattern(StructurePatternNames.LIST_PATTERN)
public interface C {
    public T get (int index);
    public int size();
    public Any set (int index) T 2);
    ...
}

Convention based on ArrayList

Write method (optional)

Arbitrary Type.

Read methods

Unconstrained Type (void or T in practice)
Read vs. Write Methods

- **Read Methods**
  - Used to get components of object
  - Getter methods
  - `size()`, `elementAt()`

- **Write Methods**
  - Used to change components of object
  - Setter methods
  - `addElement()`, `removeElement()`, `setElementAt()`
  - some used by Object Editor

- Distinction independent of conventions and important for Model-View-Controller and other paradigms you will see later

- Conventions used in Object Editor
CONVENTIONS FOR VARIABLE-SIZED COLLECTION

```java
public interface PointHistory {

    public void addElement (int x, int y);

    public Point elementAt (int index);

    public int size();

}
```

- **Write Method not recognized by OE**
- **Read Methods**
- **Arbitrary Type**
Variable-sized Collection

History

Methods added to menu associated with class

Graphic elements of dynamic collections added at their (X, Y) locations
public static void main (String[] args) {
    Map aMap = new HashMap();
    aMap.put("Nadal", 10);
    aMap.put("Federer", 17);
    aMap.put("Sampras", 14);
    System.out.println(aMap.get("Nadal"));
    System.out.println(aMap.get("nadal"));
    aMap.put("Nadal", 11);
    System.out.println(aMap.get("Nadal"));
    System.out.println(aMap);
    ObjectEditor.edit(aMap);
}
OE Conventions For Table

// associates key with value, returning last value associated with key
public <ValueType> put (<KeyType> key, <ValueType> value);
// returns last value associated with key, or null if no association
public <ValueType> get (<KeyType> key);
// optional, removes associated value, and returns it or null
public <ValueType> remove(<KeyType> key);

Necessary but not sufficient to displays all keys and elements
Displaying Stack (LIFO)

```java
public interface StringStack {
    public boolean isEmpty();
    public String getTop();
    public void push(String element);
    public void pop();
}

StringStack stringStack = new AStringStack();
stringStack.push("James Dean");
stringStack.push("Joe Doe");
stringStack.push("Jane Smith");
stringStack.push("John Smith");
ObjectEditor.edit(stringStack);
```

Does not provide read methods for reading all elements
Displaying Transparent Stack (LIFO)

```java
public interface TransparentStringStack {
    public int size();
    public String get(int index);
    public void push(String element);
    public void pop();
}
```

StringStack stringStack = new AStringStack();
stringStack.push("James Dean");
stringStack.push("Joe Doe");
stringStack.push("Jane Smith");
stringStack.push("John Smith");

Provides read methods following OE collection conventions

Can provide additional method for top value