Data Structures

- **What is a data Structure ?**
- **AssociativeArray**
- **Stack of int**
- **Stack of Objects**
- **Making AssociativeArray more generic**
- **Assgnment Discussion**

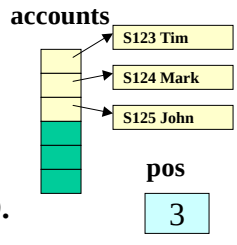
1

What is a data Structure ?

The representation of data in a structure and the associated operations.

Have we used a data structure before ?

- **Recall in the AccountsManager class we used an array of account references accounts, and an integer variable pos (to keep track of count).**
- **We used the method boolean add(Account acc) method to add a reference to an Account object which ensured accounts had unique ID.**
- **We used Account get(String ID) to get a reference to an Account(or subclass) object.**
- **Together (data + methods) is known as data structure.**



2

Is it better to separate the class ?

```
void manageCreation()
 // get user input…
 ...
 add(new Account(……))
```

```
void manageCreation()
   ...
   accounts [pos] = new Account(……);  // not safe
   pos++;
```

It is better to separate the functionality of the date structure into a separate class as in

3

```
class AssociativeArray  {
      public AssociativeArray(int n) { … }
      public boolean add(Account a) { … }
      public Account get(String ID) { … }
      public Account get(int i) { … }
      private Account[] accounts;
      private int pos;
}
```

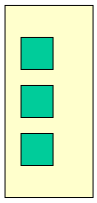
This class may then be used in AccountsManager class as in:

```
// in the AccountsManager constructor
AssociativeArray accounts = new AssociativeArray(100);

// in manageCreate() of the AccountsManager class
accounts.add(new Account("s12345","Charles",1000));

// in the manageWithdrawal()of AccountsManager class
Account a = accounts.get(“s12345”);
```

4

Data Structures - the components

- **In Object Oriented Programming we build complex applications using existing data structures (components)**
- **For example suppose in an Ecommerce application we need to ensure to keep a list of transcations, parts and accounts.**
- **If we have a class (component) say named AssociativeArray which ensures two objects in the list do not have the same ID it makes the developers job easy. We could modify the class in last slide making it more generic.**
- **The developer is not concerned with the actual implementation (it is the job of component builder - who worries about efficiency). Developer is concerned does it provide a method get(), add(), etc.**



5

Stack - a common data structure

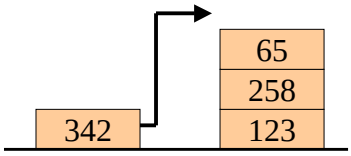
Suppose you have joined a Warehouse and you're required to write a program for managing a stack of containers.

The empty container arriving in the warehouse is automatically placed on top of a stack of containers. (Max 10)

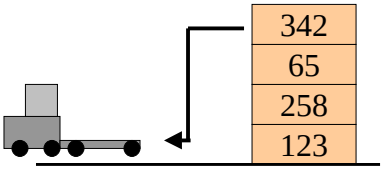
When a new container is needed for transporting goods the topmost one is retrieved.

Containers are identified by an int value, serial number.

The empty container (342) just arrived is to placed on top.



When a new container is needed the topmost one (342) is retrieved.



6

What kind of data structure is needed ?

```
Container Management System
        Store          S
        Rerieve        R
        Quit           Q
Your Input:  R
Sorry no containers available.  Try later.
Your Input: S
Enter Serial Number:   236
Container 236 stored on top
Your Input: S
Serial Number: 128
Container 128 stored on top
Your Input: S
Enter Serial Number:   239
Container 239 stored on top
Your Input: R
Container 239 is retrieved
```

```
void push(int x) {...
int void pop() { ...
boolean isEmpty(){...
boolean isFull(){...
// instance variables
// other private methods
…..
```

```
IntStack conts = new IntStack();
…
num = console.readInt();
if (conts.isFull())
  System.out.println("cannot stack now");
else cons.push(num); // stored
…
if (conts.isEmpty())
  System.out.println("no containers");
else num = cons.pop(); // retrieved
```

7

Stack Exception classes

- A StackFullException is thrown when attempting to push an element when stack is already full
- StackEmptyException is thrown when attempting to pop an element when stack is empty.

```
class StackFullException extends Exception {
  StackFullException (String errorMessage)
  {      super(errorMessage);  }
}
class StackEmptyException extends Exception{
  StackEmptyException (String errorMessage)
  {       super(errorMessage);  }
}
```

8

Implementing a Stack Class

```
class IntStack {

   IntStack ()  // constructor: stack is empty
   {      top = -1;   }

   StackFullException stackFull =
      new StackFullException("Stack is full!");
   StackEmptyException stackEmpty =
      new StackEmptyException("Stack is empty!");
   final int MAX_ELEMENTS = 10;  // max stack size
   int[] stackArray = new int[MAX_ELEMENTS];
   int top;  // keeps top of the stack

   boolean isFull() {
      return (top == MAX_ELEMENTS - 1);
   }

   boolean isEmpty()  {
      return (top == -1);
   }
```

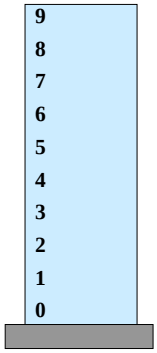
9

```
 void push(int x) throws StackFullException {
      if (isFull())
         throw stackFull;
      stackArray[++top] = x;
   }

   int pop() throws StackEmptyException {
      if(isEmpty())
         throw stackEmpty;
      return stackArray[top--];
   }

   int peek() throws StackEmptyException {
      if(isEmpty())
         throw stackEmpty;
      return stackArray[top];
   }

   void makeEmpty(){  top = -1;   }

}
```

10

Testing the Stack class

```
import java.io.*;
public class StackDriver  {
   public static void main(String[] args)  {
     IntStack myStack = new IntStack();
     try     {
        for (int i = 0; i < 10; i++)
           myStack.push (i);

        for (int i = 0; i < 10; i++)
           System.out.println(myStack.pop());
     }
     catch (StackFullException sfe)  {
        System.err.println(sfe);
     }
     catch (StackEmptyException see)  {
        System.err.println(see);
     }
   }
}
```

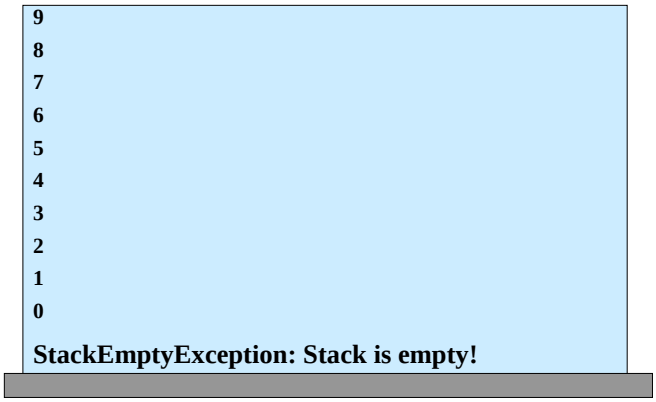


What if I attempt to pop 11 times instead of 10 ?

11

Popping one too many ...

```
for (int i = 0; i < 11; i++)
   System.out.println(myStack.pop());
```



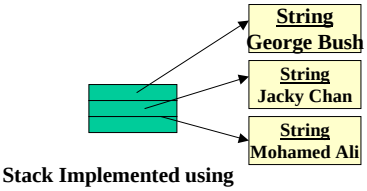
12

Generic Stack class

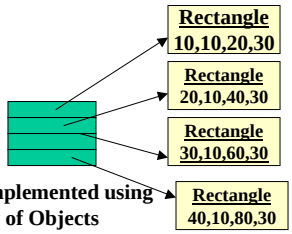
Suppose I want a Stack for keeping a number of String objects and another to keep Rectangle objects.

You may create separate Stack classes or Create a Stack using an array of Object references. Recall Object - the mother of all Java classes.

Using such a Stack class we can store any objects (not primitives)



Stack Implemented using an array of Objects



Stack Implemented using an array of Objects

13

Sample Driver using a Generic Stack class

```
import java.io.*;    import java.awt.*;
public class GenericStackDriver  {
  public static void main(String[] args)  {
     Stack myStack1 = new Stack();
     Stack myStack2 = new Stack();
    try     {
      for (int i = 0; i < 10; i++)
        myStack1.push (new String("String "+i));
      for (int i = 0; i < 10; i++)
        System.out.println(myStack1.pop());

      for (int i = 0; i < 10; i++)
        myStack2.push (new Rectangle(i*10,10,20*i,30));
      for (int i = 0; i < 10; i++)
        System.out.println(myStack2.pop());
    }
    catch (StackFullException sfe){ System.err.println(sfe);}
    catch (StackEmptyException see){System.err.println(see);}
  }
}
```

14

IntStack → Stack What are the Changes ?

Wherever you see int replace with Object as in :

```
int [] stackArray = new int[MAX_ELEMENTS];
Object[] stackArray = new Object[MAX_ELEMENTS];

void push(int x) throws StackFullException {
void push(Object x) throws StackFullException {

int pop() throws StackEmptyException {
Object pop() throws StackEmptyException {
```

15

Making the AssociativeArray more generic ...

- **Such a class (with methods add() and get()) could be useful for storing and manipulating Part, and Transaction objects as well as Account objects.**
- **But the problem is add() and get() methods makes use of the getID() method on that object - which in turn assumes that all objects (parts, accounts, trans) have a unique ID.**
- **To get around this problem we can create a more generic AssociativeArray class which assumes all objects being stored on it has a getID() method - implements the Identifiable interface with one method given below.**

```
interface Identifiable { public String getID();}
```

To store Account, Part &Transaction objects we make them implement Identifiable as in:

```
class Account implements Identifiable { …
class Part implements Identifiable { …
class Transaction implements Identifiable { …
```

16

All we have use an array of Identifiable references

- Arguments passed to add changed to Identifiable
- The returned type for get is changed to Object (or Identifiable)
- to do make our AssociativeArray more generic:

```
class AssociativeArray  {
   public AssociativeArray(int n) { … }
   public boolean add(Identifiable ident) { … }
   public Object get(String ID) { … }
   public Object get(int i) { … }
   private Identifiable[] accounts;
   private int pos;
}
```

17

```
class AssociativeArray     {
  public AssociativeArray(int n) {
       objects = new Identifiable[n];
       pos = 0;
  }
  public boolean add(Identifiable a)
       throws RepeatedKeyException       {
     if (pos == objects.length)
       throw new RuntimeException("Array Full");
              for (int i =0; i<pos; i++)
     if(a.getID().compareTo(objects[i].getID()) == 0)
         throw new RepeatedKeyException("Key Repeated"
          ,objects[i].getID(),objects[i]);
     objects[pos++] = a;
     return true;
  }

  public Object get(String ID) {
     for (int i =0; i<pos; i++)
        if(ID.compareTo(objects[i].getID()) == 0)
                    return objects[i];
        return null;
  }
```

18

```
        public Object get(int i) {
                if (i < pos) return objects[i];
                return null;
        }

        public int getSize() {     return pos;   }
        Identifiable[] objects;
        int pos;
}
```

If you want to extend the AssociativeArray class to be able to handle input output of these objects as well refer to pages 305 - 308.

19

The RepeatedKeyException

```
class RepeatedKeyException extends Exception
{
        public RepeatedKeyException(String message,
                String key , Object object) {
           super(message);
           this.object = object;
           this.key = key;
        }
        public String getKey()      { return key;      }
        public Object getObject() {       return object; }
        private       String key;
    // reference to repeated object
        private Object object;
}
```

20

Assignment Discussion

How can I get the current date ?

```
Date currentDate = new Date();
System.out.println(currentDate.toString());
```

Do I need separate classes for Transaction, Part, …

Yes.

21