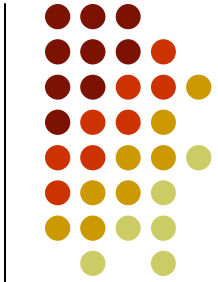
Designing an ADT



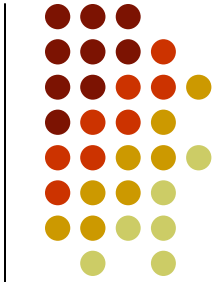
- The design of an ADT should evolve naturally during the problem-solving process
- Questions to ask when designing an ADT
  - What data does a problem require?
  - What operations does a problem require?

Examples: polynomial, appointment book

`

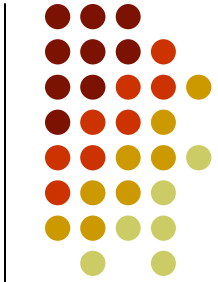
- ADT can suggest other ADTs

Appendix 1.
Java Exceptions (review)



- Exception
  - A mechanism for handling an error during execution
  - A method indicates that an error has occurred by throwing an exception

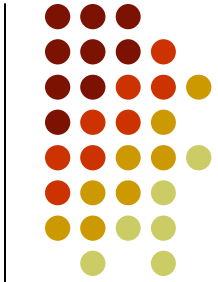
Java Exceptions



- Catching exceptions
  - `try` block
    - A statement that might throw an exception is placed within a `try` block
    - Syntax

```
try {
    statement(s);
} // end try
```

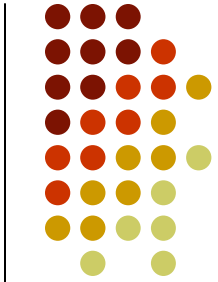
Java Exceptions



- Catching exceptions (Continued)
  - `catch` block
    - Used to catch an exception and deal with the error condition
    - Syntax

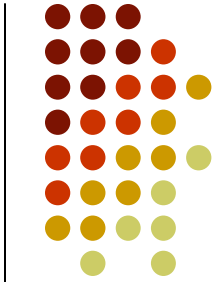
```
catch (exceptionClass identifier) {
   statement(s);
} // end catch
```

Java Exceptions



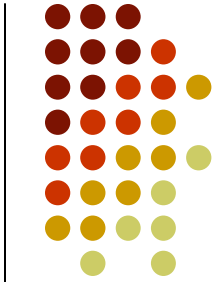
- Types of exceptions
  - Checked exceptions
    - Instances of classes that are subclasses of the `java.lang.Exception` class
    - Must be handled locally or explicitly thrown from the method
    - Used in situations where the method has encountered a serious problem

Checked exceptions



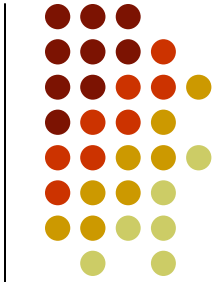
```
public class TestExceptionExample  {
  public static void getInput(String fileName) {
    FileInputStream fis;
    fis = new FileInputStream(fileName);
    // file processing code appears here
  } // end getInput

  public static void main(String[] args) {
    getInput("test.dat");
  } // end main
} // end TestExceptionExample
```

Java Exceptions



- Types of exceptions (Continued)
  - Runtime exceptions
    - Used in situations where the error is not considered as serious
    - Can often be prevented by fail-safe programming
    - Instances of classes that are subclasses of the `RuntimeException` class
    - Are not required to be caught locally or explicitly thrown again by the method

Java Exceptions



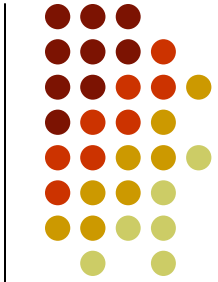
- Throwing exceptions
  - A `throw` statement is used to throw an exception

```
throw new exceptionClass
(stringArgument);
```

- Defining a new exception class
  - A programmer can define a new exception class

```
class MyException extends Exception {
  public MyException(String s) {
    super(s);
  } // end constructor
}  // end MyException
```

Implementing ADTs



- Choosing the data structure to represent the ADT's data is a part of implementation
  - Choice of a data structure depends on
    - Details of the ADT's operations
    - Context in which the operations will be used
- Implementation details should be hidden behind a wall of ADT operations
  - A program would only be able to access the data structure using the ADT operations

An Array-Based Implementation of the ADT List

- An array-based implementation
  - A list's items are stored in an array `items`
  - A natural choice
    - Both an array and a list identify their items by number
  - A list's k^{th} item will be stored in `items[k-1]`

An Array-Based Implementation of the ADT List

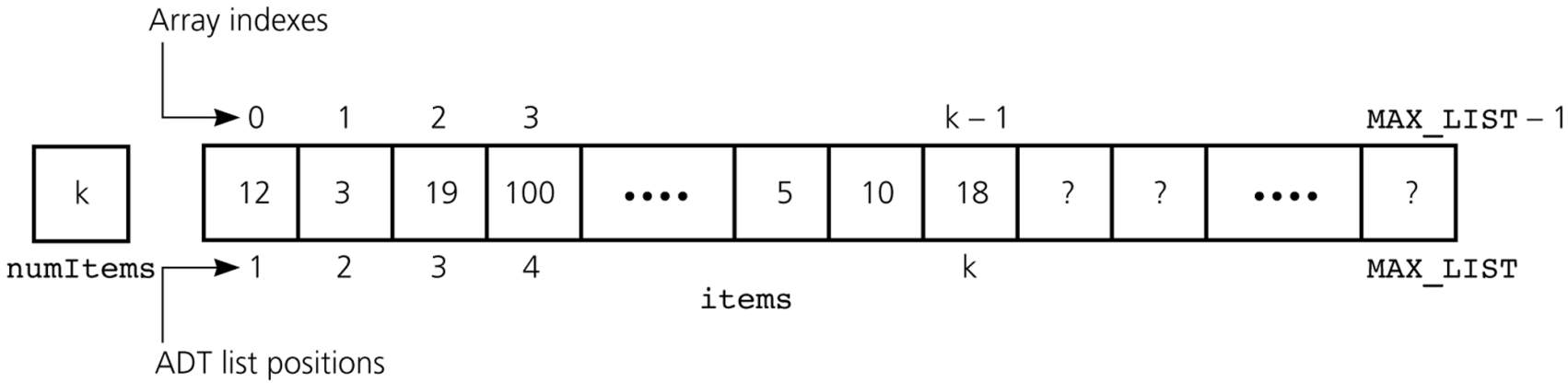


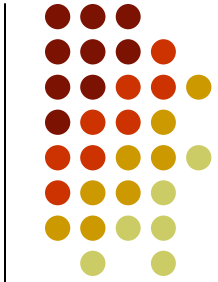
Figure 4-11

An array-based implementation of the ADT list

An Array-Based Implementation of the ADT List

```
public class ListArrayBased
implements ListInterface {

  private static final int MAX_LIST = 50;
  private Object items[];
    // an array of list items
  private int numItems;
    // number of items in list
```

Appendix 2.
Arrays in Java (review)

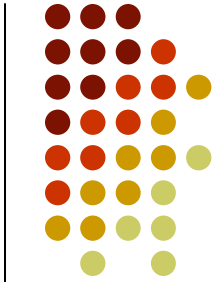


- Arrays are sequences of identically typed values
- Values are stored at specific numbered positions in the array
  - The first value is stored at index 0, the second at index 1, the *i*th at index *i*-1, and so on
  - The last item is stored at position *n*-1, assuming that *n* values are stored in the array
- Values are stored sequentially in main memory

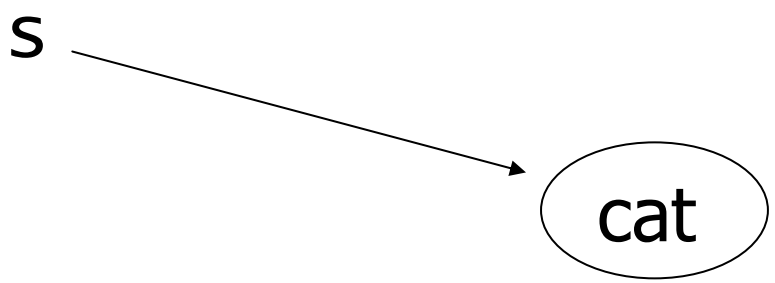
Arrays in Java

- To declare an array follow the type with (empty) `[]`s
  - `int[] grade; //or`
  - `int grade[]; //both declare an int array`
- In Java arrays are objects!

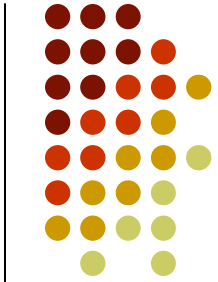
Objects in Java



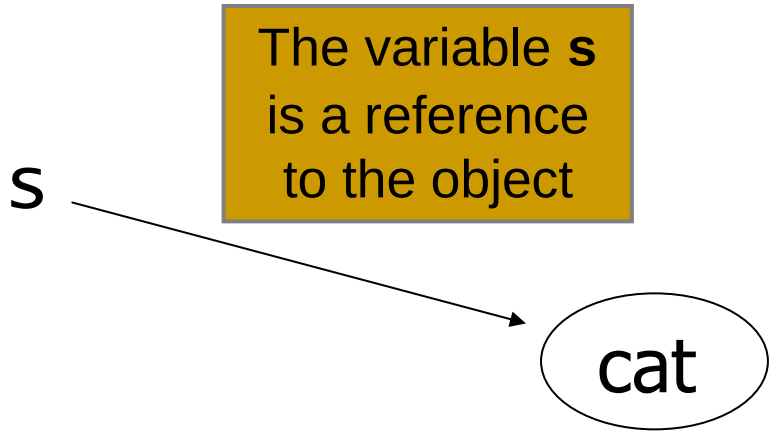
```
String s = new String("cat");
```



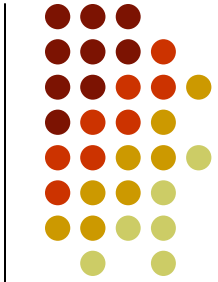
Objects in Java (review)



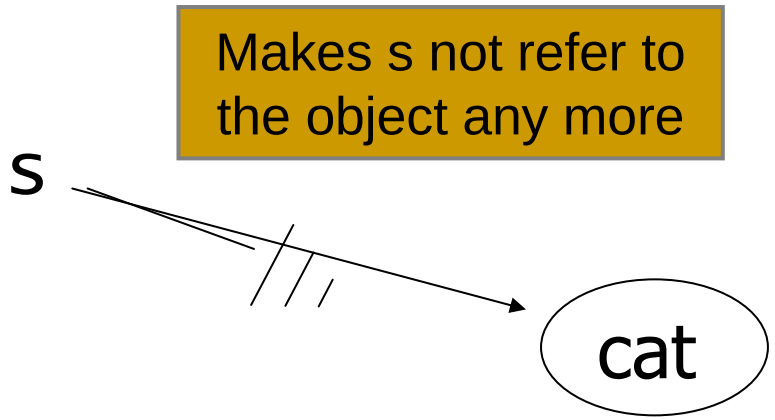
```
String s = new String("cat");
```



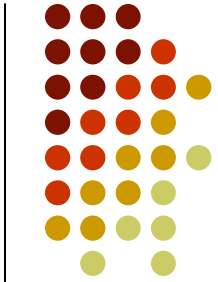
Objects in Java



```
String s = new String("cat");
S = null;
```



Objects in Java

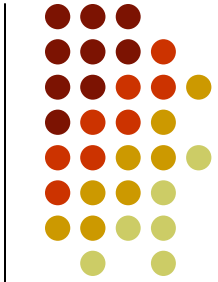


```
String s = new String("cat");
S = null;
```

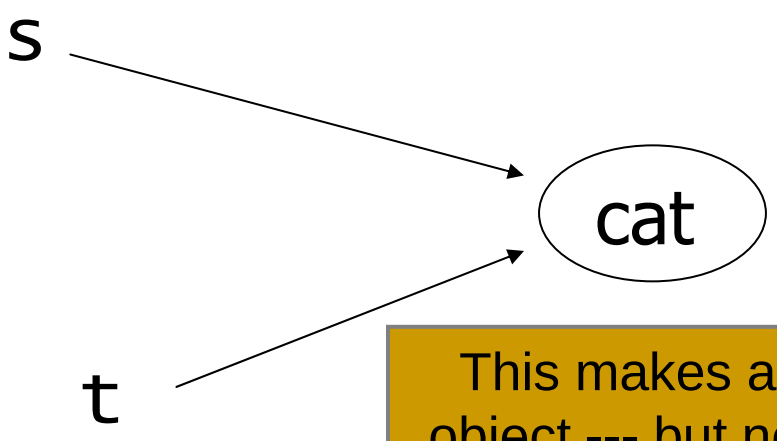
Makes s not refer to the object any more

The object gets deleted by Java's automated garbage collection

Objects in Java



```
String s = new String("cat");
String t = s;
```

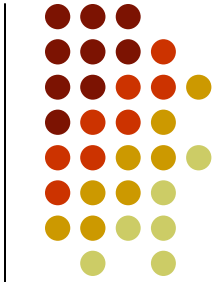


This makes another reference to the object --- but no new object is created!!

Arrays in Java

- To declare an array follow the type with (empty) `[]`s
  - `int[] grade; //or`
  - `int grade[]; //both declare an int array`
- In Java arrays are objects so must be created with the **`new`** keyword
  - To create an array of ten integers:
    - `int[] grade = new int[10];`
    - Note that the array size has to be specified, although it can be specified with a variable at run-time

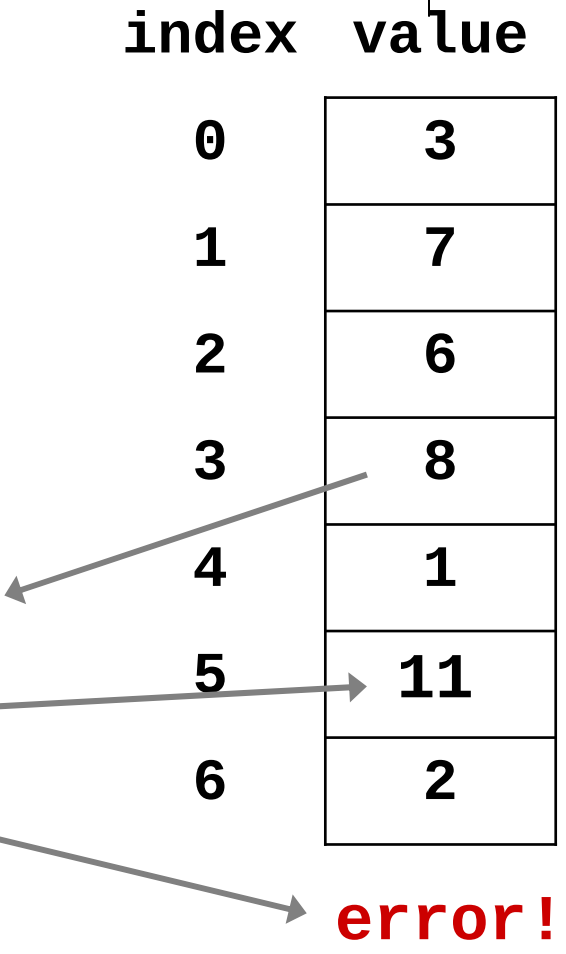
Arrays in Java



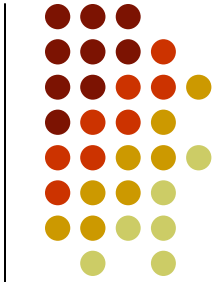
- When the array is created memory is reserved for its contents
- Initialization lists can be used to specify the initial values of an array, in which case the **new** operator is not used
  - `int[] grade = {87, 93, 35}; //array of 3 ints`
- To find the length of an array use its **.length** variable
  - `int numGrades = grade.length; //note: not .length()!!`

Array Indexing

- `int[] arr = {3,7,6,8,1,7,2};` creates a new integer array with seven elements
  - The elements are assigned values as given in the initialization list
- Individual elements can be accessed by referring to the array name and the appropriate index
  - `int x = arr[3];` would assign the value of the fourth array element (8) to `x`
  - `arr[5] = 11;` would change the sixth element of the array from 7 to 11
  - `arr[7] = 3;` would result in an error because the index is out of bounds



Arrays and Main Memory

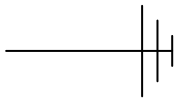


```
int[] grade;
```

Declares an array variable

main memory is depicted below

grade



"null pointer"

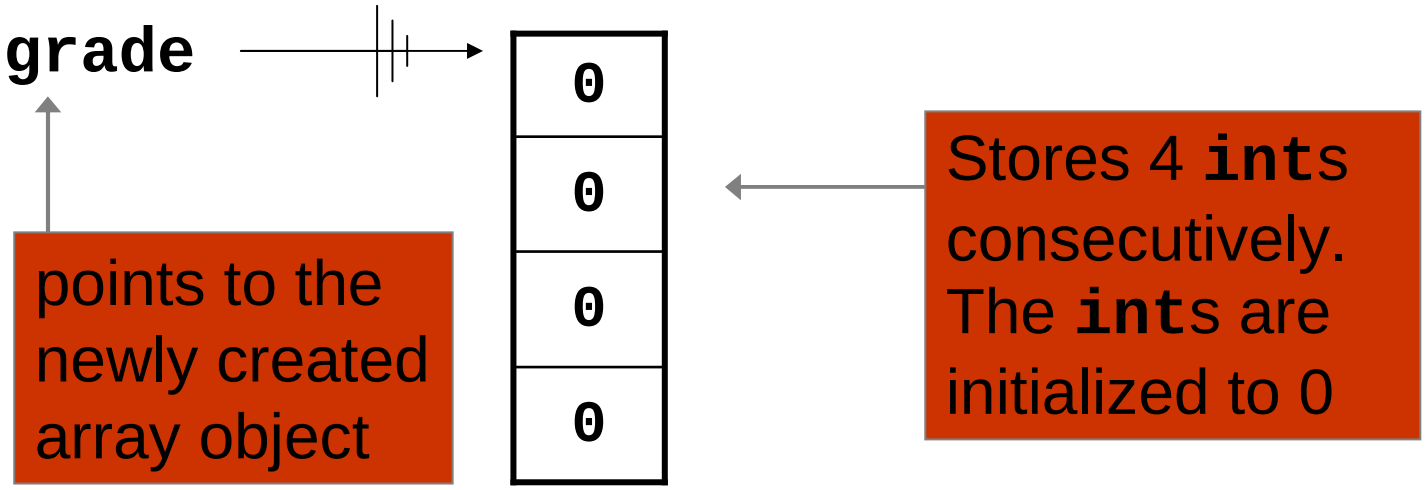
In Java arrays are objects, so this is a reference variable

Arrays and Main Memory

```
int[] grade;
grade = new int[4];
```

Creates a new array of size 4

main memory is depicted below

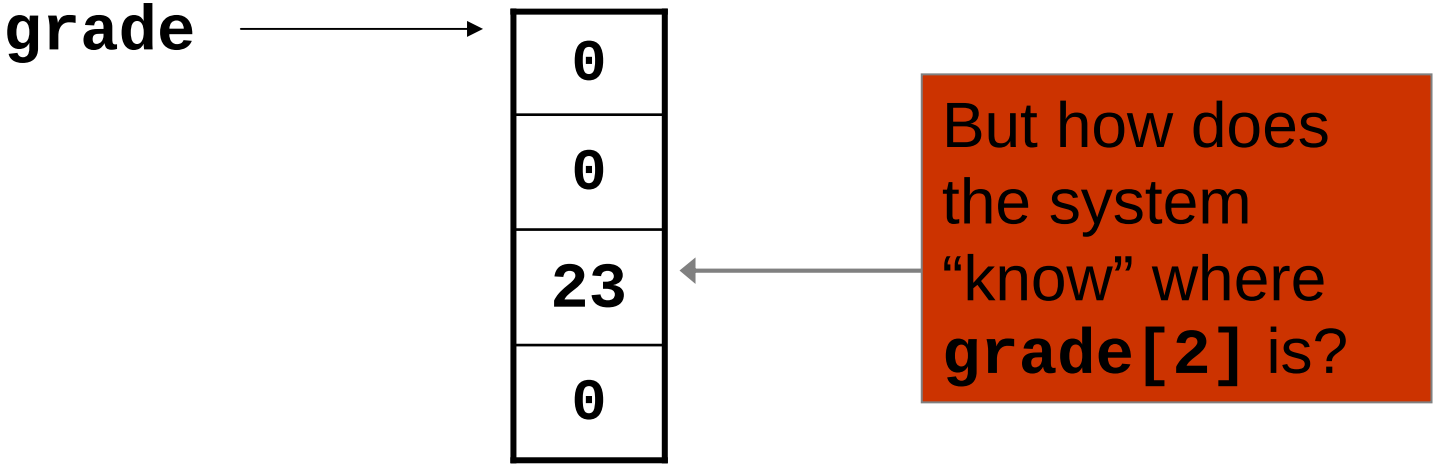


Arrays and Main Memory

```
int[] grade;
grade = new int[4];
grade[2] = 23;
```

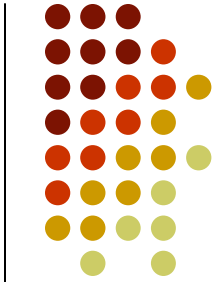
Assigns 23 to the third item in the array

main memory is depicted below



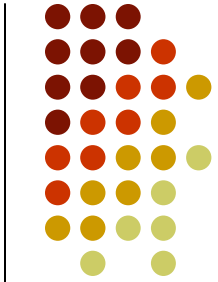
But how does the system "know" where **`grade[2]`** is?

Offset Calculations



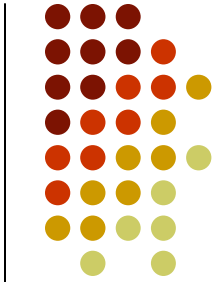
- Given something like **`grade[2] = 23;`** how do we find a particular element in the array?
- We know the address of the first element in the array
- Because we know the type of the values stored in the array, we know the size of each element in the array
  - 4 bytes in the case of an **`int`**
- We know which element we want to access
- We can therefore calculate the address of the desired element as being:
  - ***address of first element*** + ***index*** * ***size of stored type***

Passing Arrays to Methods



- Array variables are reference variables
  - When an array variable is passed as an argument to a method the method is being given the address of an array object
  - Not a new copy of the array object
- Any changes made to the array in the method are therefore made to the original (and only) array object
  - If this is not desired, a copy of the array should be made within the method

Arrays are *Static* Data Structures



- The size of an array must be specified when it is created with **new** ***and cannot be changed***
- If the array is full new items can't be added to it
  - There are, time consuming, ways around this
  - To avoid this problem make arrays much larger than they are needed
  - However this wastes space