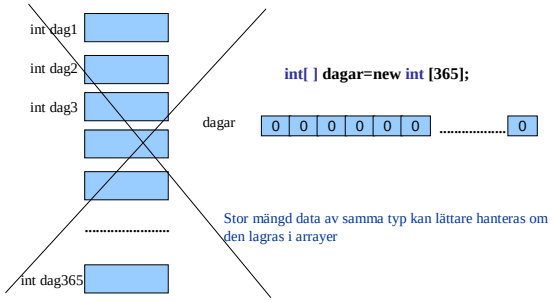
Arrays

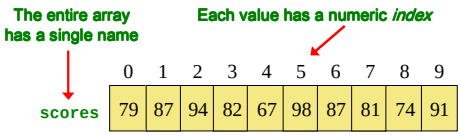
- Arrays are objects that help us organize large amounts of information
- Chapter 7 focuses on:
  - array declaration and use
  - bounds checking and capacity
  - arrays that store object references
  - variable length parameter lists
  - multidimensional arrays
  - the `ArrayList` class
  - polygons and polylines
  - mouse events and keyboard events

Vi behöver ett program som sammlar in data för tempervärden under ett år



Arrays

- An *array* is an ordered list of values



An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

```
scores[2]
```

  refers to the value 94 (the 3rd value in the array)
- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Arrays

- For example, an array element can be assigned a value, printed, or used in a calculation:

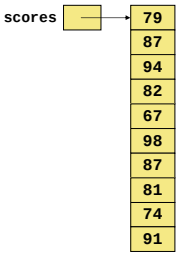
```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

System.out.println ("Top = " + scores[5]);
```


Arrays

- The values held in an array are called *array elements*
- An array stores multiple values of the same type – the *element type*
- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, an array of characters, an array of `String` objects, an array of `Coin` objects, etc.
- In Java, the array itself is an object that must be instantiated

Arrays

- Another way to depict the `scores` array:



Declaring Arrays

- The `scores` array could be declared as follows:

```
int[] scores = new int[10];
```

- The type of the variable `scores` is `int[]` (an array of integers)
- Note that the array type does not specify its size, but each object of that type has a specific size
- The reference variable `scores` is set to a new array object that can hold 10 integers

Declaring Arrays

- Some other examples of array declarations:

```
float[] prices = new float[500];

boolean[] flags;
flags = new boolean[20];

char[] codes = new char[1750];
```


Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in range 0 to N-1
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called automatic *bounds checking*

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99
- If the value of `count` is 100, then the following reference will cause an exception to be thrown:

```
System.out.println (codes[count]);
```

- It's common to introduce *off-by-one errors* when using arrays

problem

```
for (int index=0; index <= 100; index++)
   codes[index] = index*50 + epsilon;
```


Bounds Checking

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name:

```
scores.length
```

- Note that `length` holds the number of elements, not the largest index
- See `ReverseOrder.java`
- See `LetterCount.java`

Alternate Array Syntax

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following two declarations are equivalent:

```
float[] prices;
float prices[];
```

- The first format generally is more readable and should be used

Initializer Lists

- An *initializer list* can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};

char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```


Initializer Lists

- Note that when an initializer list is used:
  - the new operator is not used
  - no size value is specified
- The size of the array is determined by the number of items in the initializer list
- An initializer list can be used only in the array declaration
- See `Primes.java`

Arrays as Parameters

- An entire array can be passed as a parameter to a method
- Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other
- Therefore, changing an array element within the method changes the original
- An individual array element can be passed to a method as well, in which case the type of the formal parameter is the same as the element type

Arrays of Objects

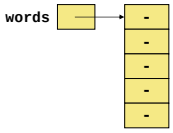
- The elements of an array can be object references
- The following declaration reserves space to store 5 references to `String` objects

```
String[] words = new String[5];
```

- It does NOT create the `String` objects themselves
- Initially an array of objects holds `null` references
- Each object stored in an array must be instantiated separately

Arrays of Objects

- The `words` array when initially declared:

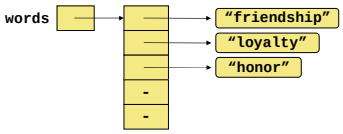


At this point, the following reference would throw a `NullPointerException`:

```
System.out.println (words[0]);
```


Arrays of Objects

- After some `String` objects are created and stored in the array:



Arrays of Objects

- Keep in mind that `String` objects can be created using literals
- The following declaration creates an array object called `verbs` and fills it with four `String` objects created using string literals

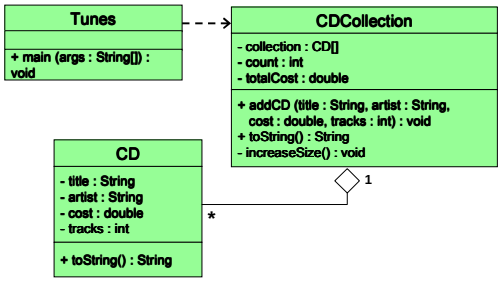
```
String[] verbs = {"play", "work", "eat", "sleep"};
```

Arrays of Objects

- The following example creates an array of `Grade` objects, each with a string representation and a numeric lower bound
- See `GradeRange.java`
- See `Grade.java`
- Now let's look at an example that manages a collection of `CD` objects
- See `Tunes.java`
- See `CDCollection.java`
- See `CD.java`

Arrays of Objects

- A UML diagram for the `Tunes` program:



Command-Line Arguments

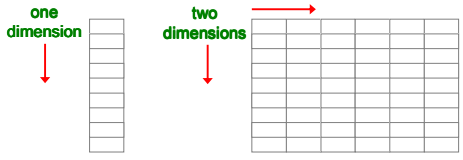
- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter
- These values come from *command-line arguments* that are provided when the interpreter is invoked
- For example, the following invocation of the interpreter passes three `String` objects into `main`:

```
> java StateEval pennsylvania texas arizona
```

- These strings are stored at indexes 0-2 of the array parameter of the `main` method
- See `NameTag.java`

Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns



Two-Dimensional Arrays

- To be precise, in Java a two-dimensional array is an array of arrays
- A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] scores = new int[12][50];
```

- A array element is referenced using two index values:

```
value = scores[3][6]
```

- The array stored in one row can be specified using one index

Two-Dimensional Arrays

Expression	Type	Description
`table`	`int[][]`	2D array of integers, or array of integer arrays
`table[5]`	`int[]`	array of integers
`table[5][12]`	`int`	integer

See `TwoDArray.java`

See `SodaSurvey.java`

The ArrayList Class

- The `ArrayList` class is part of the `java.util` package
- Like an array, it can store a list of values and reference each one using a numeric index
- However, you cannot use the bracket syntax with an `ArrayList` object
- Furthermore, an `ArrayList` object grows and shrinks as needed, adjusting its capacity as necessary

The ArrayList Class

- Elements can be inserted or removed with a single method invocation
- When an element is inserted, the other elements "move aside" to make room
- Likewise, when an element is removed, the list "collapses" to close the gap
- The indexes of the elements adjust accordingly

The ArrayList Class

- An `ArrayList` stores references to the `Object` class, which allows it to store any kind of object
- See `Beatles.java`
- We can also define an `ArrayList` object to accept a particular type of object
- The following declaration creates an `ArrayList` object that only stores `Family` objects

```
ArrayList<Family> reunion = new ArrayList<Family>
```

- This is an example of *generics*, which are discussed further in Chapter 12

ArrayList Efficiency

- The `ArrayList` class is implemented using an underlying array
- The array is manipulated so that indexes remain continuous as elements are added or removed
- If elements are added to and removed from the end of the list, this processing is fairly efficient
- But as elements are inserted and removed from the front or middle of the list, the remaining elements are shifted

Polygons and Polylines

- Arrays can be helpful in graphics processing
- For example, they can be used to store a list of coordinates
- A *polygon* is a multisided, closed shape
- A *polyline* is similar to a polygon except that its endpoints do not meet, and it cannot be filled
- See `Rocket.java`
- See `RocketPanel.java`

The Polygon Class

- The `Polygon` class can also be used to define and draw a polygon
- It is part of the `java.awt` pacakage
- Versions of the overloaded `drawPolygon` and `fillPolygon` methods take a single `Polygon` object as a parameter instead of arrays of coordinates
- A `Polygon` object encapsulates the coordinates of the polygon

Mouse Events

- Events related to the mouse are separated into *mouse events* and *mouse motion events*
- Mouse Events:

mouse pressed	the mouse button is pressed down
mouse released	the mouse button is released
mouse clicked	the mouse button is pressed down and released without moving the mouse in between
mouse entered	the mouse pointer is moved onto (over) a component
mouse exited	the mouse pointer is moved off of a component

Mouse Events

- Mouse Motion Events:

mouse moved	the mouse is moved
mouse dragged	the mouse is moved while the mouse button is pressed down

Listeners for mouse events are created using the `MouseListener` and `MouseMotionListener` interfaces

A `MouseEvent` object is passed to the appropriate method when a mouse event occurs

Mouse Events

- For a given program, we may only care about one or two mouse events
- To satisfy the implementation of a listener interface, empty methods must be provided for unused events
- See `Dots.java`
- See `DotsPanel.java`

Mouse Events

- *Rubberbanding* is the visual effect in which a shape is "stretched" as it is drawn using the mouse
- The following example continually redraws a line as the mouse is dragged
- See `RubberLines.java`
- See `RubberLinesPanel.java`

Key Events

- A *key event* is generated when the user types on the keyboard

key pressed	a key on the keyboard is pressed down
key released	a key on the keyboard is released
key typed	a key on the keyboard is pressed down and released

Listeners for key events are created by implementing the `KeyListener` interface

A `KeyEvent` object is passed to the appropriate method when a key event occurs

Key Events

- The component that generates a key event is the one that has the current *keyboard focus*
- Constants in the `KeyEvent` class can be used to determine which key was pressed
- The following example "moves" an image of an arrow as the user types the keyboard arrow keys
- See `Direction.java`
- See `DirectionPanel.java`