

## “super” gives us all the necessary methods

getMessage( ) - what the user typed that generated the error

toString( ) - the computer's description of the message

printStackTrace( ) - list all the methods from main, to the one that generated the error

---

---

---

---

---

---

---

---

## 2. throwing

```
public void increasePowerLevel() throws
    highTempException
{
    powerLevel = powerLevel + 10;
    temperature = powerLevel * 100;
    if (temperature >= 10000)
    {
        throw new highTempException();
    }
} // end increasePower
```

---

---

---

---

---

---

---

---

## 3. using (i.e. catching)

```
try
{
    chernobyl.increasePowerLevel();
}
catch (highTempException hte)
{
    hte.printStackTrace();
    System.exit(0);
}
```

---

---

---

---

---

---

---

---

## Array example

```
int x;  
int z[] = new int[10]; // integer too complicated,  
                      // needs "new", z[0] to z[9]  
for (x = 0; x < 10; x = x+1)  
{  
    z[x] = x;  
}
```

What is z[4]? 4, in fact...  
z[0] = 0 z[1] = 1 z[2] = 2 ... to z[9] = 9

---

---

---

---

---

---

---

---

## Arrays

- A difficult concept at first : `int z[5];`
- Means: `z[0]`, `z[1]`, `z[2]`, `z[3]`, `z[4]` are separate integer variables
- Starts at 0.
- An array with 5 elements is *indexed* from 0 to 4.
- in Java: `int z[] = new int[5];`

---

---

---

---

---

---

---

---

## Arrays do not have to be numbers

```
int x;  
String myArray[] = new String[5];  
myArray[0] = "hello";  
myArray[1] = "we";  
myArray[2] = "have";  
myArray[3] = "a quiz";  
myArray[4] = "on Monday";  
for (x=0; x <= 4; x++)  
{  
    System.out.println( myArray[x] );  
}
```

---

---

---

---

---

---

---

---

## Sorting - “selection” Sort

- the array is full (no empty slots)
- the element to be placed in each slot, is located and exchanged with current occupant

---

---

---

---

---

---

---

---

## String JNC helper methods

```
String userInput; // userInput is an object of
                  // String Java Native Class
if ( userInput.compareTo("a") == 0 )
    // .compareTo is a method that returns a
    // zero if string object == string in ("...")
    // Why return an integer?
    // -1..-26 if userInput precedes it alphabetically
    // +1..+26 if userInput is alphabetically after
```

---

---

---

---

---

---

---

---

## String JNC helper methods

```
if (userInput.compareToIgnoreCase("X") == 0 )
    // returns zero if equal... return type is an
    // int

if (userInput.equalsIgnoreCase("X") == true )
    // returns true if equal... return type is a
    // boolean
```

---

---

---

---

---

---

---

---

```
import java.util.Arrays;
```

```
Arrays.sort ( a );
```

```
// will sort in ascending order
```

“a[ ]” can be Strings, integers, whatever

---

---

---

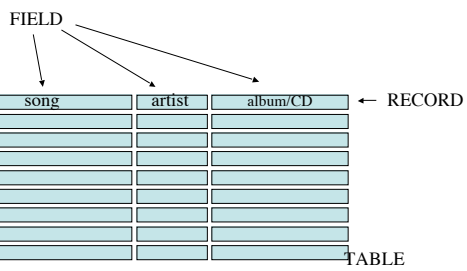
---

---

---

---

---



---

---

---

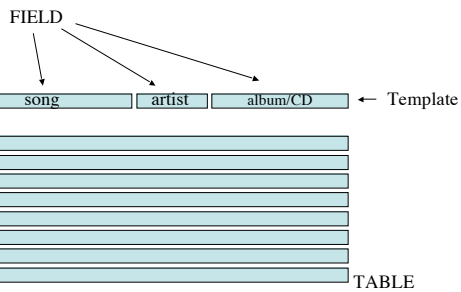
---

---

---

---

---



---

---

---

---

---

---

---

---

a musicDatabase

```
// an array of musicRecords  
  
musicRecord musicDatabase[ ] =  
new musicRecord[ numberOfSongs ];
```

---

---

---

---

---

---

---

create a TreeMap, instead of an array

```
TreeMap myTable = new TreeMap();  
for (x=0; x<numberOfSongs; x++)  
{  
    myTable.put(  
        musicDatabase[x].getSong(),  
        musicDatabase[x] );  
}  
// (key field, record )
```

---

---

---

---

---

---

---

use the key field

```
musicRecord myRecord =  
(musicRecord) myTable.get("EboLorama");  
  
// note cast
```

---

---

---

---

---

---

---

## The List Interface

- A List, is an array, of indefinite size
- It grows and shrinks as you add and remove elements
- Two types of lists:
  - ArrayList
  - LinkedList
- `import java.util.*;`

---

---

---

---

---

---

---

---

## Two “types”?

- An ArrayList is a general purpose array, but with list properties
  - certain methods
  - grows and shrinks
- A LinkedList contains “mappings” stored with each element which virtually sort the array (without moving elements around).
  - remember `placeInSongList` and `placeInArtistList`?

---

---

---

---

---

---

---

---

Two lists:

|       |            |            |   |   |
|-------|------------|------------|---|---|
| index |            |            |   |   |
| 0     | AC / DC    | AC / DC    |   | 1 |
| 1     | Beatles    | Beatles    | 0 | 4 |
| 2     | Incubus    | Incubus    | 4 | 3 |
| 3     | Ozzy       | Ozzy       | 2 | 5 |
| 4     | Eagles     | Eagles     | 1 | 2 |
| 5     | Pink Floyd | Pink Floyd | 3 |   |

---

---

---

---

---

---

---

---

## A general-purpose interface

```
List myList;
```

*decide later in the code which kind of list*

```
myList = new ArrayList();
```

or

```
myList = new LinkedList();
```

---

---

---

---

---

---

---

---

## some methods

To find the current size of a list we use the *size* method.

To empty a list use the *clear* method.

---

---

---

---

---

---

---

---

## add( ) and get( )

- add elements with add( )
- get elements with get( ) - casting required

---

---

---

---

---

---

---

---

```
List sample = new ArrayList();
sample.add("Hello");
sample.add(14);
```

```
> javac tryStuff.java
cannot resolve symbol sample.add( int );
```

---

---

---

---

---

---

---

---

```
List sample = new ArrayList();
sample.add("Hello");
// sample is now committed
sample.add("another String");
```

---

---

---

---

---

---

---

---

### index-based methods

- **add**(int index, Object element)  
Inserts the specified element at the specified position in this list
- **set**(int index, Object element)  
Replaces the element at the specified position in this list with the specified element.
- **get**(int index)  
Returns the element at the specified position in this list

---

---

---

---

---

---

---

---

### the iterator

- traverses the list using *next()* and *hasNext()* methods
- all Lists have an iterator, just go get it

```
List myList;  
myList = new ArrayList();  
Iterator itr = myList.iterator();  
while ( itr.hasNext () )  
{ ...
```

---

---

---

---

---

---

---

### the Linked List

- has INTERNAL previous and next fields
- you don't see them
- can't manipulate them
- allows internal speedup of searching and sorting

```
List yourList = new ArrayList();  
List myList = new LinkedList();
```

myList is faster

---

---

---

---

---

---

---

### Announcements

- Quiz on Wed 11/17
- No class on Monday 11/22

---

---

---

---

---

---

---

## Arrays vs. Lists

Arrays:

```
[ 1033 ]
```

```
[ 714 ]
```

```
[ 2102 ]
```

```
[ 64 ]
```

or

```
[ "cat" ]
```

```
[ "rain" ]
```

```
[ "George" ]
```

Lists:

```
[ name ][ phone ][ address ][ age ]
```

```
[ name ][ phone ][ address ][ age ]
```

```
[ name ][ phone ][ address ][ age ]
```

```
[ name ][ phone ][ address ][ age ]
```

---

---

---

---

---

---

---

---

## the Collection class

- manages Lists (ArrayLists and LinkedLists)
- offers a very useful method:

```
.sort( List );
```

but.... how can you “sort” a “list” of “objects”

---

---

---

---

---


---

---

---

## well, sometimes it’s easy

```
List myList = new ArrayList( );  
for (x=1000; x > 0; x = x-1)  
{  
  myList.add( x ); // 1000, 999, 998, 997, etc  
}  
Collections.sort( myList );  
Collections.sort( myList, Collections.reverseOrder() );
```

 what's this?

---

---

---

---

---

---

---

---

that second parameter....

`Collections.sort ( List );`

`Collections.sort ( List, Comparator );`

---

---

---

---

---

---

---

### the Comparator Interface

- allows you to design a custom comparison (like, comparing music records by song).

```
import java.util.Comparator;
public class songComparator implements Comparator
{
    public int compare( Object obj1, Object obj2 )
    {
        musicRecord mr1, mr2;

        mr1 = (musicRecord)obj1;
        mr2 = (musicRecord)obj2;
        return ( mr1.getSong().compareTo( mr2.getSong() ) );
    }
}
```

---

---

---

---

---

---

---

### the comparator interface

- must have a method called “compare”
- which takes two objects
- and returns an integer that reflects “greater than” relationship, similar to “compareTo” method in String

---

---

---

---

---

---

---

now, use it with any list

```
List musicDatabase = new ArrayList();  
musicDatabase.add( new musicRecord( ...  
musicDatabase.add( new musicRecord( ...  
musicDatabase.add( new musicRecord( ...  
musicDatabase.add( new musicRecord( ...  
  
songComparator sc = new songComparator();  
  
Collections.sort( musicDatabase, sc );
```

---

---

---

---

---

---

---

---

### SC

```
import java.util.Comparator;  
public class songComparator implements Comparator  
{  
    public int compare( Object obj1, Object obj2 )  
    {  
        musicRecord mr1, mr2;  
        String song1, song2;  
        mr1 = (musicRecord)obj1;  
        mr2 = (musicRecord)obj2;  
        // musicRecord has a getSng() method that returns a  
        // String, so use the String.compareTo() method  
        song1 = mr1.getSng();  
        song2 = mr2.getSng();  
        return ( song1.compareTo( song2 ) );  
        //note: return (mr1.getSng().compareTo(mr2.getSng()));  
    }  
}
```

---

---

---

---

---

---

---

---

Threads  
read Chapter 9  
(sorry... not Chapter 10)

---

---

---

---

---

---

---

---

## what are threads?

- independently running programs within one Unix process
- can communicate through static variables that they share
- define the “Client / Server” relationship
- used when a task needs undivided attention
  - playing mp3s
  - running a task while waiting for user input

---

---

---

---

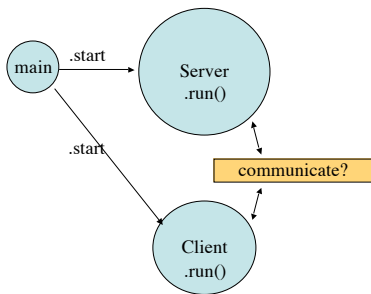
---

---

---

---

## how threads operate



---

---

---

---

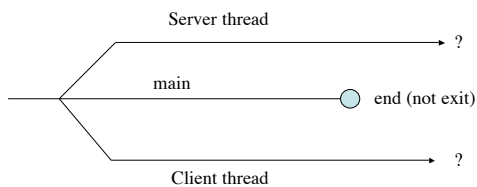
---

---

---

---

## timeline



---

---

---

---

---

---

---

---

## writing a client thread

```
public class ClientClass extends Thread
{
public void run()
{
System.out.println("Starting Client");
do while
{
// stuff
} (true);
}
}
```

---

---

---

---

---

---

---

---

## writing a Server

```
public class ServerClass extends Thread {
public void run() {
while {
// accept commands here
} (true);
}
public static void main (String [] args) {
ClientClass clientThread = new ClientClass();
clientThread.start();
ServerClass serverThread = new ServerClass();
serverThread.start();
} // end main
} // end class
```

---

---

---

---

---

---

---

---

## talk to each other

```
public class commClass
{
public static String ClientToServer = "idle";
public static String ServerToClient = "idle";
} // end commClass
```

- static means everyone can use the Strings
  - commClass.ClientToServer = "exit";
- defines a "protocol"

---

---

---

---

---

---

---

---

### some useful methods

`Thread.sleep( milliseconds )` - suspends this thread's activity

---

---

---

---

---

---

---

---

### stopping a thread

- a clean end to a threads `.run()` method ends that thread.
- `System.exit(0)` from ANYWHERE ends all threads

---

---

---

---

---

---

---

---

### Base 2 (Binary)

14652 - illegal

$$10110101 = 1x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 =$$

$$128 + 0 + 32 + 16 + 0 + 4 + 0 + 1 = 181_{10}$$

---

---

---

---

---

---

---

---

### Base 16 - Hexadecimal

- Harder, because it's > 10
- $A_{16} = 10_{10}$   $B_{16} = 11_{10}$   $C_{16} = 12_{10}$   $D_{16} = 13_{10}$   $E_{16} = 14_{10}$   $F_{16} = 15_{10}$

$$1AE2_{16} = 1 \times 16^3 + 10 \times 16^2 + 14 \times 16^1 + 2 \times 1$$

$$= 4096 + 2560 + 224 + 2$$
$$= 6882_{10}$$

---

---

---

---

---

---

---

---

### A Byte

- Eight bits
- Can represent a number from 0 to  $255_{10}$
- Reflect computer memory as a huge bank of 8-bit switches
  
- Switch is ON = 1
- Switch is OFF = 0

---

---

---

---

---

---

---

---

### New Java type

```
public static byte x;
```

means that x can have a value from (-127) to +127

---

---

---

---

---

---

---

---

## Integer.toString

does decimal to other number system conversion,  
returns a string

```
Integer.toString( number, 2);  
// returns binary string
```

```
Integer.toString( number, 16);  
// returns hex string
```

---

---

---

---

---

---

---

## dec - bin - hex

|              |               |
|--------------|---------------|
| 0000 - 0 - 0 | 1000 - 8 - 8  |
| 0001 - 1 - 1 | 1001 - 9 - 9  |
| 0010 - 2 - 2 | 1010 - 10 - A |
| 0011 - 3 - 3 | 1011 - 11 - B |
| 0100 - 4 - 4 | 1100 - 12 - C |
| 0101 - 5 - 5 | 1101 - 13 - D |
| 0110 - 6 - 6 | 1110 - 14 - E |
| 0111 - 7 - 7 | 1111 - 15 - F |

---

---

---

---

---

---

---

## Converting Binary to Hex

1010011110011110101100110100

step 1: break into "nibbles"

1010 0111 1001 1110 1011 0011 0100

step 2: convert each to Hex number

1010 0111 1001 1110 1011 0011 0100

A 7 9 E B 3 4

---

---

---

---

---

---

---

## Converting Hex to Binary

26D47FE<sub>16</sub>

2 6 D 4 7 F E

0010 0110 1101 0100 0111 1111 1110

---

---

---

---

---

---

---

---

## modulo

- useful for converting decimal to binary. *Modulo* is nothing more than "remainder after division."
- So 20 modulo 5 is 0 because 20 divided by 5 is 4 with no remainder.

21 modulo 5 is 1

22 modulo 5 is 2

23 modulo 5 is 3

24 modulo 5 is 4

25 modulo 5 is 0

In Java, modulo is represented as the percent sign. So `int a = 20 % 5 ;` sets a to be zero.

---

---

---

---

---

---

---

---

## Bits/Bytes can represent anything

Letters (called ASCII) - American Standard Code for Information Interchange

A - 65 - 1000 0001

B - 66 - 1000 0010

C - 67 - 1000 0011

D - 68 - 1000 0100

a - 97 - 1100 0001

b - 98 - 1100 0010

c - 99 - 1100 0011

d - 100 - 1100 0100

---

---

---

---

---

---

---

---

**Java Support for Binary**  
“Bitwise” – operate on separate  
binary digits

Bitwise operators are of two  
types:

- shift operators
- boolean operators

---

---

---

---

---

---

---

---

**Bit Position**

0xBC  
1 0 1 1 1 1 0 0  
position: 7 6 5 4 3 2 1 0  
MSB                  LSB  
most                  least  
significant          significant  
bit                    bit

---

---

---

---

---

---

---

---

**Shifting Bits**

The shift operators are used to shift the  
binary digits of an integer number to the  
right or the left.

```
byte i = 13; // is 0000 1101  
i = i << 2; // is 0011 0100 = 5210  
i = i >> 3; // is 0000 0110 = 610
```

---

---

---

---

---

---

---

---

## Bitwise AND

remember True AND True = True ?

$$0 \& 0 = 0$$

$$0 \& 1 = 0$$

$$1 \& 0 = 0$$

$$1 \& 1 = 1$$

---

---

---

---

---

---

---

---

## Bitwise OR

$$0 \mid 0 = 0$$

$$0 \mid 1 = 1$$

$$1 \mid 0 = 1$$

$$1 \mid 1 = 1$$

---

---

---

---

---

---

---

---

## A byte at a time...

|                 |                 |
|-----------------|-----------------|
| 1111 1111       | 1111 1111       |
| &               |                 |
| 0011 0010       | 0000 0000       |
| ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ | ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ |
| 0011 0010       | 1111 1111       |

---

---

---

---

---

---

---

---

## Manipulating Bytes

```
byte i = 0x5A; // 0101 1010  
byte j = 0x0F; // 0000 1111  
byte k;
```

```
k = i | j; // 0101 1111 = 0x5F = 9510
```

```
k = i & j; // 0000 1010 = 0x0A = 1010
```

---

---

---

---

---

---

---

---

## Good Exam Questions

```
byte x = 0xAA;  
    1010 1010
```

```
what is bit 0?  
    0
```

```
what is the MSB?  
    1
```

```
what is bit 3?  
    1
```

---

---

---

---

---

---

---

---

## Exam Questions - Bitwise Operations

```
byte x = 0x03; // 0000 0011  
byte y = 0xFA; // 1111 1010  
byte q,r;
```

```
q = x | y; what is q?  
    1111 1011
```

```
r = x & y; what is r?  
    0000 0010
```

---

---

---

---

---

---

---

---

### A "MASK"

- A byte with just one 1, and all 0's
- Mask for bit 1: 0000 0010
- Mask for bit 7: 1000 0000
- Mask for bit 6: 0100 0000

---

---

---

---

---

---

---

---

### Masking (isolate a bit)

```
byte x = 0xBC; // 1011 1100  
byte y = 0x62 // 0110 0010  
byte mask = 0x40; // 0100 0000 (mask bit 6)
```

```
r = x & mask;  
0000 0000 (r is zero, therefore bit 6 of x is 0)  
q = y & mask;  
0100 0000 (q is nonzero, bit 6 of y must be 1)
```

---

---

---

---

---

---

---

---

### Good Exam Questions

1. convert  $1ACC_{16}$  to decimal
2. convert  $12AE_{16}$  to binary
3. convert 0111 1011 to hexadecimal
4. convert 0111 1011 to decimal
5. what is  $13_{10}$  shifted right two bits?
6. what is  $0 \& 1$ ?
7. what is  $1 \mid 1$ ?

---

---

---

---

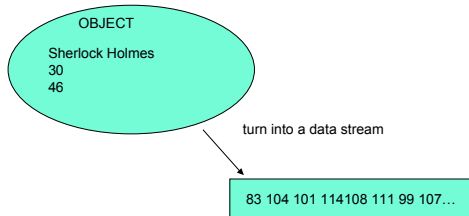
---

---

---

---

## Serializable - means “in a row”



---

---

---

---

---

---

---

---

## the Serializable Interface

- no methods are involved, so you don't have to write anything
- called a “tagging” interface, because Java just adds variables to your object, to keep the place of *your* variables in a data stream
- Objects that are serializable can be used by “streaming classes”: file I/O, network I/O

---

---

---

---

---

---

---

---

## to and from streams

- to a file:

```
FileOutputStream target = new FileOutputStream("filename.ser");
ObjectOutput out = new ObjectOutputStream( target );
out.writeObject( anyObject );
out.close();
```

- from a file:

```
FileInputStream source = new FileInputStream( "filename.ser" );
ObjectInputStream in = newObjectInputStream( source );
gridTestSer buttons = (gridTestSer) in.readObject();
in.close();
```

---

---

---

---

---

---

---

---

## using a byte array

- to a byte array

```
ByteArrayOutputStream target = new ByteArrayOutputStream();  
ObjectOutputStream out = new ObjectOutputStream( target );  
out.writeObject(anyObject);  
byte[] buf = target.toByteArray();  
out.close();
```

- from a byte array

```
ByteArrayInputStream source = new ByteArrayInputStream( buf );  
ObjectInputStream in = new ObjectInputStream( source );  
gridTestSer buttons = (gridTestSer) ois.readObject();  
in.close();
```

---

---

---

---

---

---

---

---

## additional topics

- benchmarking,
- timestamping, Gregorian Calendar
- Singleton design pattern
- networking / protocol
- error logging

---

---

---

---

---

---

---

---