
CSC220 (CSI)

Computational Problem Solving

Using Classes and Objects

The College of New Jersey

Please turn off your cell phone!

Data Types

- **Data type.** Set of **values** and **operations** on those values.
- **Primitive types.** Ops directly translate to machine instructions.

Data type	Value set	Operations
boolean	true, false	not, and, or, xor
int	-2^{31} to $2^{31} - 1$	add, subtract, multiply
double	any of 2^{64} possible reals	add, subtract, multiply

- **Object.** Holds a data type value; variable name refers to object. It is defined by **class**.
- **Impact.** Enables us to create our own data types; define operations on them; and integrate into our programs.

Creating Objects

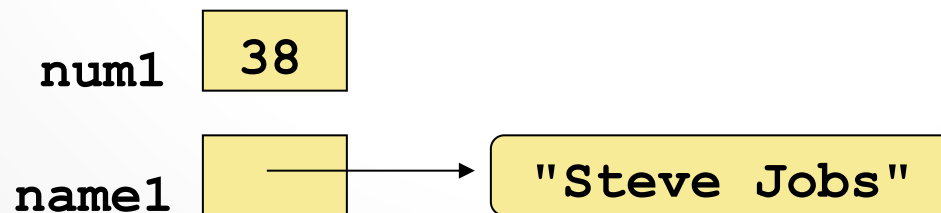
- A variable holds either a primitive value or a **reference** to an object
- A class name can be used as a type to declare an **object reference variable**

`String title; //No object is created with this declaration.`

- The object itself must be created separately

`//calls the String constructor, a special method that sets up the object`
`title = new String("Java Software Solutions");`

- An object reference variable holds the address of an object



Creating Objects

- Generally, we use the new operator to create an object
- Creating an object is called **instantiation**
- An object is an **instance** of a particular class

Declare a variable (object **name**)

Call a constructor to create an **object**

```
String sentence;  
Sentence = new String("Hello World!");  
System.out.println( sentence.length( ) )
```

Object name

Call a method that
Works on the object's value

Invoking Methods

- We've seen that once an object has been instantiated, we can use the **dot operator** to invoke its methods

```
numChars = title.length()
```

- A method may **return a value**, which can be used in an assignment or expression
- A method invocation can be thought of as asking an object to perform a service

Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

Before:

num1	38
num2	96

```
num2 = num1;
```

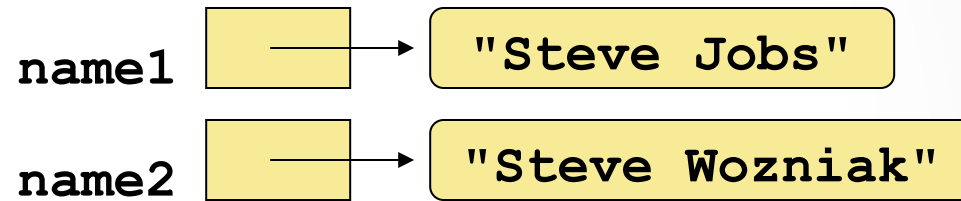
After:

num1	38
num2	38

Reference Assignment

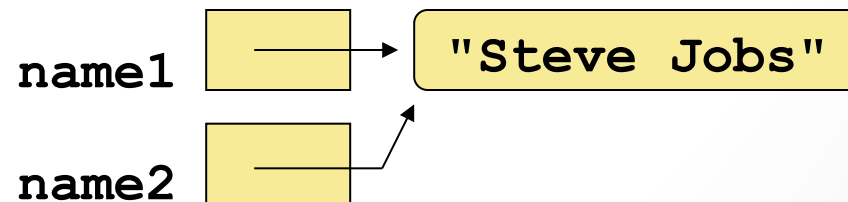
- For object references, assignment copies the address:

Before:



`name2 = name1;`

After:



Aliases

- Two or more references that refer to the same object are called **aliases** of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program
- The object is useless, and therefore is called *garbage*
- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection

The String Class

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

- This is special syntax that works only for strings
- Each string literal (enclosed in double quotes) represents a `String` object

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed
- Therefore we say that an object of the `String` class is `immutable`
- To refer to a particular character within a string, we specify the character's numeric `index`
- The indexes begin at zero in each string

String Indexes

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4
- See `StringMutation.java`

```
//*****
```

Output

Original string: "Change is inevitable"

Length of string: 20

Mutation #1: Change is inevitable, except from vending machines.

Mutation #2: CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.

Mutation #3: CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.

Mutation #4: NGX IS INXVITABLX, XXCXPT F

Mutated length: 27

```
System.out.println("Original string: \"\" + phrase + "\"");  
System.out.println("Length of string: \" + phrase.length());
```

```
mutation1 = phrase.concat(", except from vending machines.");  
mutation2 = mutation1.toUpperCase();  
mutation3 = mutation2.replace('E', 'X');  
mutation4 = mutation3.substring(3, 30);
```

```
// Print each mutated string
```

```
System.out.println("Mutation #1: \" + mutation1);  
System.out.println("Mutation #2: \" + mutation2);  
System.out.println("Mutation #3: \" + mutation3);  
System.out.println("Mutation #4: \" + mutation4);  
System.out.println("Mutated length: \" + mutation4.length());
```

```
}
```

```
}
```

Quick Check

What output is produced by the following?

```
String str = "Space, the final frontier.";
System.out.println(str.length());
System.out.println(str.substring(7));
System.out.println(str.toUpperCase());
System.out.println(str.length());
```

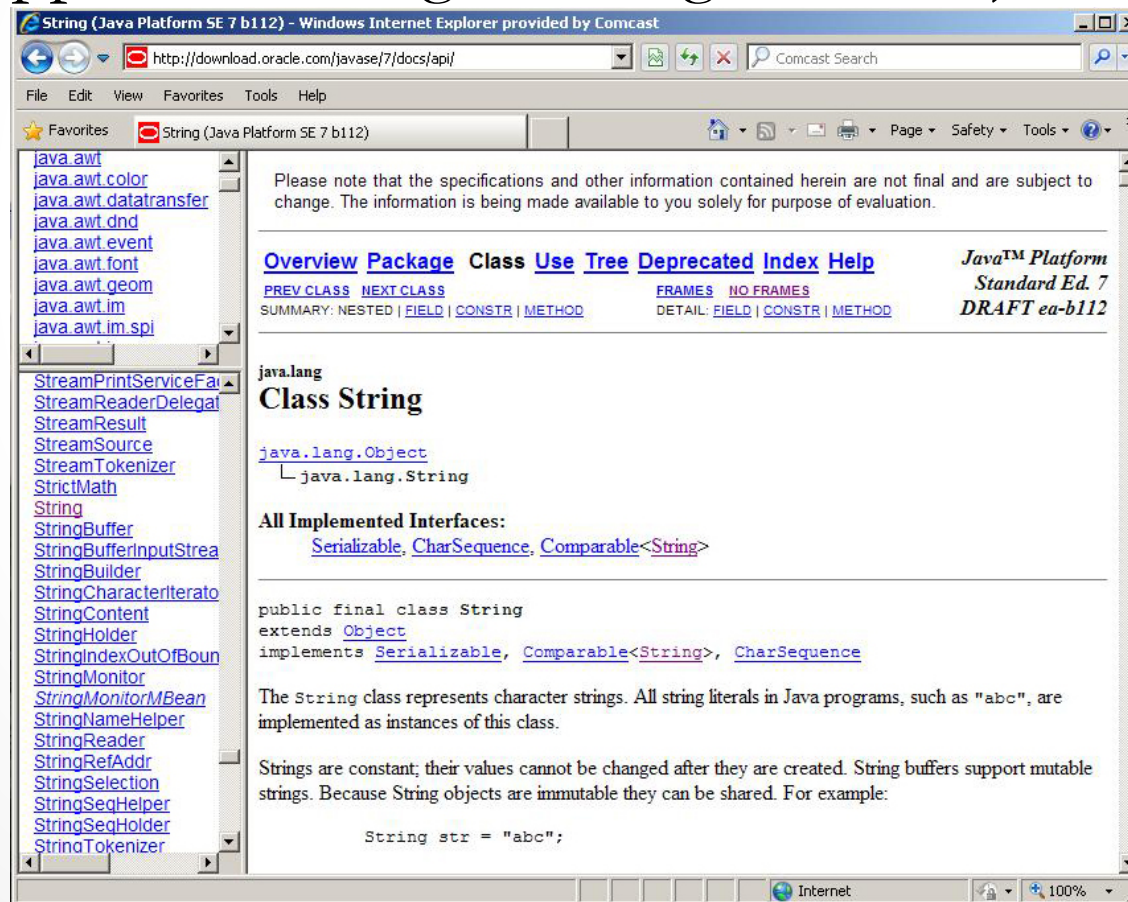
```
26
the final frontier.
SPACE, THE FINAL FRONTIER.
26
```

Class Libraries

- A **class library** is a collection of classes that we can use when developing programs
- The **Java standard class library** is part of any Java development environment
- Its classes are not part of the Java language per se, but we rely on them heavily
- Various classes we've already used (`System`, `Scanner`, `String`) are part of the Java standard class library

The Java API

- The Java class library is sometimes referred to as the Java API (Application Programming Interface)



Packages

- For purposes of accessing them, classes in the Java API are organized into *packages*
- These often overlap with specific APIs
- Examples:

Package

Purpose

java.lang

General support

java.applet

Creating applets for the web

java.awt

Graphics and graphical user interfaces

javax.swing

Additional graphics capabilities

java.net

Network communication

java.util

Utilities

javax.xml.parsers

XML document processing

The import Declaration

- When you want to use a class from a package, you could use its **fully qualified name**

```
java.util.Scanner
```

- Or you can **import** the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

The import Declaration

- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line:

```
import java.lang.*;
```

- That's why we didn't have to import the `System` or `String` classes explicitly in earlier programs
- The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported

The Random Class

- The Random class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A Random object performs complicated calculations based on a **seed value** to produce a stream of seemingly random values
- See `RandomNumbers.java`

```

//*****
// RandomNumbers.java          Author: Lewis/Loftus
//
// Demonstrates the creation of pseudo-random numbers using the
// Random class.
//*****

import java.util.Random;

public class RandomNumbers
{
    //-----
    // Generates random numbers in various ranges.
    //-----
    public static void main(String[] args)
    {
        Random generator = new Random();
        int num1;
        float num2;

        num1 = generator.nextInt();
        System.out.println("A random integer: " + num1);

        num1 = generator.nextInt(10);
        System.out.println("From 0 to 9: " + num1);
    }
}

```

continued

continued

```
num1 = generator.nextInt(10) + 1;
System.out.println("From 1 to 10: " + num1);

num1 = generator.nextInt(15) + 20;
System.out.println("From 20 to 34: " + num1);

num1 = generator.nextInt(20) - 10;
System.out.println("From -10 to 9: " + num1);

num2 = generator.nextFloat();
System.out.println("A random float (between 0-1): " + num2);

num2 = generator.nextFloat() * 6; // 0.0 to 5.999999
num1 = (int)num2 + 1;
System.out.println("From 1 to 6: " + num1);
    }
}
```

continued

Sample Run

```
num1 A random integer: 672981683
Syst From 0 to 9: 0
      From 1 to 10: 3
num1 From 20 to 34: 30
Syst From -10 to 9: -4
num1 A random float (between 0-1): 0.18538326
Syst From 1 to 6: 3
```

```
num2 = generator.nextFloat();
System.out.println("A random float (between 0-1): " + num2);

num2 = generator.nextFloat() * 6; // 0.0 to 5.999999
num1 = (int)num2 + 1;
System.out.println("From 1 to 6: " + num1);
}
}
```

Quick Check

Given a `Random` object named `gen`, what range of values are produced by the following expressions?

`gen.nextInt(25)`

Range

0 to 24

`gen.nextInt(6) + 1`

1 to 6

`gen.nextInt(100) + 10`

10 to 109

`gen.nextInt(50) + 100`

100 to 149

`gen.nextInt(10) - 5`

-5 to 4

`gen.nextInt(22) + 12`

12 to 33

Quick Check

Write an expression that produces a random integer in the following ranges:

<u>Range</u>	
0 to 12	<code>gen.nextInt(13)</code>
1 to 20	<code>gen.nextInt(20) + 1</code>
15 to 20	<code>gen.nextInt(6) + 15</code>
-10 to 0	<code>gen.nextInt(11) - 10</code>

The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
 - absolute value
 - square root
 - exponentiation
 - trigonometric functions

The Math Class

- The methods of the `Math` class are **static methods** (also called **class methods**)
- Static methods are invoked through the class name – no object of the `Math` class is needed

```
value = Math.cos(90) + Math.sqrt(delta);
```

- See `Quadratic.java`

```
//*****  
// Quadratic.java  
//
```

```
// Demonstrates the use  
// based on user input.  
//*****
```

```
import java.util.Scanner;  
public class Quadratic{
```

```
//-----
```

```
// Determines the roots of a quadratic equation.
```

```
//-----
```

```
public static void main(String[] args){
```

```
    int a, b, c; //  $ax^2 + bx + c$ 
```

```
    double discriminant, root1, root2;
```

```
    Scanner scan = new Scanner(System.in);
```

```
    System.out.print("Enter the coefficient of x squared: ");
```

```
    a = scan.nextInt();
```

```
    System.out.print("Enter the coefficient of x: ");
```

```
    b = scan.nextInt();
```

```
    System.out.print("Enter the constant: ");
```

```
    c = scan.nextInt();
```

```
    // Use the quadratic formula to compute the roots.
```

```
    discriminant = Math.pow(b, 2) - (4 * a * c);
```

```
    root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
```

```
    root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);
```

```
    System.out.println("Root #1: " + root1);
```

```
    System.out.println("Root #2: " + root2);
```

```
}
```

```
}
```

Sample Run

Enter the coefficient of x squared: 3

Enter the coefficient of x: 8

Enter the constant: 4

Root #1: -0.6666666666666666

Root #2: -2.0

Formatting Output

- It is often necessary to format output values in certain ways so that they can be presented properly
- The Java standard class library contains classes that provide formatting capabilities
- The `NumberFormat` class allows you to format values as currency or percentages
- The `DecimalFormat` class allows you to format values based on a pattern
- Both are part of the `java.text` package

Formatting Output

- The `NumberFormat` class has static methods that return a formatter object

`getCurrencyInstance()`

`getPercentInstance()`

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format
- See `Purchase.java`

```

//*****
//  Purchase.java          Author: Lewis/Loftus
//
//  Demonstrates the use of the NumberFormat class to format output.
//*****

import java.util.Scanner;
import java.text.NumberFormat;

public class Purchase
{
    //-----
    //  Calculates the final price of a purchased item using values
    //  entered by the user.
    //-----
    public static void main(String[] args)
    {
        final double TAX_RATE = 0.06;  // 6% sales tax

        int quantity;
        double subtotal, tax, totalCost, unitPrice;

        Scanner scan = new Scanner(System.in);

```

continued

continued

```
NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
NumberFormat fmt2 = NumberFormat.getPercentInstance();

System.out.print("Enter the quantity: ");
quantity = scan.nextInt();

System.out.print("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
System.out.println("Subtotal: " + fmt1.format(subtotal));
System.out.println("Tax: " + fmt1.format(tax) + " at "
                  + fmt2.format(TAX_RATE));
System.out.println("Total: " + fmt1.format(totalCost));
    }
}
```


continued

Sample Run

```
NumberFormat f
NumberFormat f
Enter the quantity: 5
Enter the unit price: 3.87
Subtotal: $19.35
Tax: $1.16 at 6%
Total: $20.51

System.out.print("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
System.out.println("Subtotal: " + fmt1.format(subtotal));
System.out.println("Tax: " + fmt1.format(tax) + " at "
                    + fmt2.format(TAX_RATE));
System.out.println("Total: " + fmt1.format(totalCost));
}
}
```

Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in various ways
- For example, you can specify that the number should be truncated to three decimal places
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number
- See `CircleStats.java`

```

//*****
//  CircleStats.java      Auth
//
//  Demonstrates the formatting
//  DecimalFormat class.
//*****
import java.util.Scanner;
import java.text.DecimalFormat;
public class CircleStats{
    //-----
    //  Calculates the area and circumference of a circle given its
    //  radius.
    //-----
    public static void main(String[] args){
        int radius;
        double area, circumference;
        Scanner scan = new Scanner(System.in);
        System.out.print ("Enter the circle's radius: ");
        radius = scan.nextInt();
        area = Math.PI * Math.pow(radius, 2);
        circumference = 2 * Math.PI * radius;
        // Round the output to three decimal places
        DecimalFormat fmt = new DecimalFormat ("0.###");

        System.out.println ("The circle's area: " + fmt.format(area));
        System.out.println ("The circle's circumference: "
                           + fmt.format(circumference));
    }
}

```

Sample Run

Enter the circle's radius: 5
 The circle's area: 78.54
 The circle's circumference: 31.416

Enumerated Types

- Java allows you to define an **enumerated type**, which can then be used to declare variables
- An enumerated type declaration lists all possible values for a variable of that type
- The values are identifiers of your own choosing
- The following declaration creates an enumerated type called Season

```
enum Season {winter, spring, summer, fall};
```

- Any number of values can be listed

Enumerated Types

- Once a type is defined, a variable of that type can be declared:

```
Season time;
```

- And it can be assigned a value:

```
time = Season.fall;
```

- The values are referenced through the name of the type
- Enumerated types are **type-safe** – you cannot assign any value other than those listed

Ordinal Values

- Internally, each value of an enumerated type is stored as an integer, called its **ordinal value**
- The first value in an enumerated type has an ordinal value of zero, the second one, and so on
- However, you cannot assign a numeric value to an enumerated type, even if it corresponds to a valid ordinal value

Enumerated Types

- The declaration of an enumerated type is a special type of class, and each variable of that type is an object
- The `ordinal` method returns the ordinal value of the object
- The `name` method returns the name of the identifier corresponding to the object's value
- See `IceCream.java`

```

//*****
//  IceCream.java          Author: Lewis/Loftus
//
//  Demonstrates the use of enumerated types
//*****
public class IceCream {
    enum Flavor {vanilla, chocolate, strawberry,
                 rockyRoad, mintChocolateChip}

    //-----
    //  Creates and uses variables of the Flavor enum
    //-----
    public static void main (String[] args){
        Flavor cone1, cone2, cone3;
        cone1 = Flavor.rockyRoad;
        cone2 = Flavor.chocolate;
        System.out.println("cone1 value: " + cone1);
        System.out.println("cone1 ordinal: " + cone1.ordinal());
        System.out.println("cone1 name: " + cone1.name());
        System.out.println();
        System.out.println("cone2 value: " + cone2);
        System.out.println("cone2 ordinal: " + cone2.ordinal());
        System.out.println("cone2 name: " + cone2.name());
        cone3 = cone1;
        System.out.println();
        System.out.println("cone3 value: " + cone3);
        System.out.println("cone3 ordinal: " + cone3.ordinal());
        System.out.println("cone3 name: " + cone3.name());
    }
}

```

Output

```

cone1 value: rockyRoad
cone1 ordinal: 5
cone1 name: rockyRoad
cone2 value: chocolate
cone2 ordinal: 1
cone2 name: chocolate
cone3 value: rockyRoad
cone3 ordinal: 5
cone3 name: rockyRoad

```


Wrapper Classes

- The `java.lang` package contains **wrapper classes** that correspond to each primitive type:

<u>Primitive Type</u>	<u>Wrapper Class</u>
<code>byte</code>	<code>Byte</code>
<code>short</code>	<code>Short</code>
<code>int</code>	<code>Integer</code>
<code>long</code>	<code>Long</code>
<code>float</code>	<code>Float</code>
<code>double</code>	<code>Double</code>
<code>char</code>	<code>Character</code>
<code>boolean</code>	<code>Boolean</code>

Wrapper Classes

- The following declaration creates an `Integer` object which represents the integer 40 as an object

```
Integer age = new Integer(40);
```

- Wrapper classes contain static methods that help manage the associated type
 - For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:

```
num = Integer.parseInt(str);
```
- They often contain useful constants as well
 - For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values

Autoboxing

- **Autoboxing** is the automatic conversion of a primitive value to a corresponding wrapper object:

```
Integer obj;  
int num = 42;  
obj = num;
```

- The assignment creates the appropriate `Integer` object
- The reverse conversion (called **unboxing**) also occurs automatically as needed

Quick Check

Are the following assignments valid? Explain.

Double value = 15.75;

Yes. The double literal is autoboxed into a Double object.

Character ch = new Character('T');
char myChar = ch;

Yes, the char in the object is unboxed before the assignment.

The ArrayList Class

- An `ArrayList` object stores a list of objects, and is often processed using a loop
- The `ArrayList` class is part of the `java.util` package
- You can reference each object in the list using a numeric index
- An `ArrayList` object grows and shrinks as needed, adjusting its capacity as necessary

The ArrayList Class

- Index values of an `ArrayList` begin at 0 (not 1):

0	"Bashful"
1	"Sleepy"
2	"Happy"
3	"Dopey"
4	"Doc"

- Elements can be inserted and removed
- The indexes of the elements adjust accordingly

ArrayList Methods

- Some ArrayList methods:

`boolean add(E obj)`

`void add(int index, E obj)`

`Object remove(int index)`

`Object get(int index)`

`boolean isEmpty()`

`int size()`

The ArrayList Class

- The type of object stored in the list is established when the `ArrayList` object is created:

```
ArrayList<String> names = new ArrayList<String>();
```

```
ArrayList<Book> list = new ArrayList<Book>();
```

- This makes use of Java *generics*, which provide additional type checking at compile time
- An `ArrayList` object cannot store primitive types, but that's what wrapper classes are for
- See `Beatles.java`


```

//*****
//  Beatles.java      Author: Lewis/Loftus
//  Demonstrates the use of a ArrayList object.
//*****
import java.util.ArrayList;
public class Beatles{
    //-----
    //  Stores and modifies a list of band members
    //-----
    public static void main(String[] args){
        ArrayList<String> band = new ArrayList<String>();
        band.add("Paul");
        band.add("Pete");
        band.add("John");
        band.add("George");
        System.out.println(band);
        int location = band.indexOf("Pete");
        band.remove(location);
        System.out.println(band);
        System.out.println("At index 1: " + band.get(1));
        band.add(2, "Ringo");
        System.out.println("Size of the band: " + band.size());
        int index = 0;
        while (index < band.size()){
            System.out.println(band.get(index));
            index++;
        }
    }
}

```

Output

```

[Paul, Pete, John, George]
[Paul, John, George]
At index 1: John
Size of the band: 4
Paul
John
Ringo
George

```