[DB 5-2] 11th Mar. 2023

Consider it very seriously

Java Programming

(Quick Review)

Spring, 2023

Jaewook Byun

Ph.D., Assistant Professor, Department of Software, Sejong University

jwbyun@sejong.ac.kr

https://sites.google.com/view/jack-dfpl/home

https://www.youtube.com/channel/UC988e-Y8nto0LXVae0aqaOQ

Topics

- Basic Programming
  - Output
  - Expression
  - Input
  - Operator
  - Branch
  - Loop
  - Array
  - Method
- Object-oriented Programming
  - Class
  - Constructor
  - Getters/Setters
  - Inheritance
  - Upcasting/Downcasting
  - Method Overriding
  - Generic and Polymorphism
  - Interface

Output

- Print out a message on a console
  - Using println(String x)

void java.io.PrintStream.println(String x)

Prints a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

Parameters:
x The `String` to be printed.

```
public class P1 {
        public static void main(String[] args) {
                System.out.println("Hello Java");
        }
}
```

  - Make System's console print out a message

- Auto-formatting: Ctrl+Shift+f
- Hint: Ctrl+Space

Expression

- Look into the following statement

```
String message = "Hello Java";
```

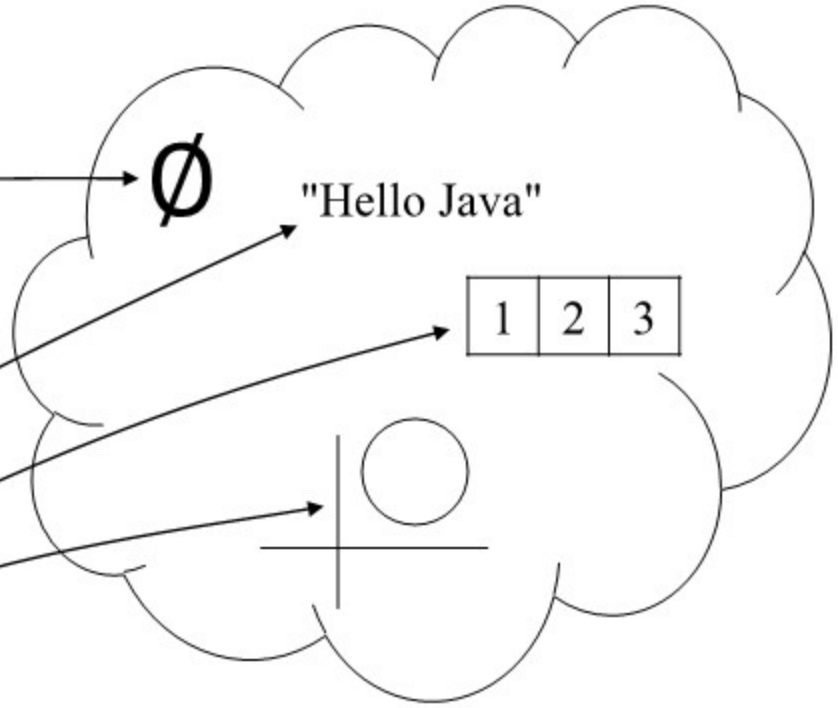
- Assign a string value, "Hello Java", to a variable called 'message' of String type

Data Type	Identifier	Operator	Literal	end of statement
String	message	=	"Hello Java"	;

Expression

Data Type	Identifier	Operator	Literal	end of statement
String	message	=	"Hello Java"	;

- Literal
  - Integer Literal
    - Decimal number: e.g., 15;
    - Hexadecimal number: e.g., 0x15; // start with 0x
  - Long Literal
    - e.g., 24L; // start with L or l
  - Real-value Literal
    - e.g., 0.1234f; // start with f or F
  - Character Literal
    - e.g., 'w'
    - e.g., '\t' // there are special characters
  - Logical-value Literal
    - e.g., true, false
  - Null-value Literal
    - i.e., null // means 'empty'
  - Reference-value Literal
    - String reference Literal: e.g., "Hello Java"
    - Array reference Literal: e.g., new int[]{1,2,3};
    - Class-instance reference Literal: e.g., new Circle(1,2,3);



Expression

Data Type	Identifier	Operator	Literal	end of statement
String	message	=	"Hello Java"	;

- Data Type
  - Primitive Type
    - boolean
    - char
    - byte
    - short
    - int
    - long
    - float
    - double
  - Reference Type
    - Array
    - Class instance
    - Interface

- Literal
  - Integer Literal
    - Decimal number: e.g., 15;
    - Hexadecimal number: e.g., 0x15;// start with 0x
  - Long Literal
    - e.g., 24L; // start with L or l
  - Real-value Literal
    - e.g., 0.1234f; // start with f or F
  - Character Literal
    - e.g., 'w'
    - e.g., '\t' // there are special characters
  - Logical-value Literal
    - e.g., true, false
  - Null-value Literal
    - i.e., null // means 'empty'
  - Reference-value Literal
    - String reference Literal: e.g., "Hello Java"
    - Array reference Literal: e.g., new int[]{1,2,3};
    - Class-instance reference Literal: e.g., new Circle(1,2,3);

Expression

Data Type	Identifier	Operator	Literal	end of statement
String	message	=	"Hello Java"	;

- Identifier
  - keep a specific size of memory space and name it
  - e.g.,
    - int intValue;
    - long longValue;
    - float floatValue;
    - char charValue;
    - boolean boolValue;

    - String stringValue;
    - int[] intArray;
    - Circle circleValue;

Expression

Data Type	Identifier	Operator	Literal	end of statement
String	message	=	"Hello Java"	;

- Operator
  - manipulate a certain value or operand
- Assignment operator (=)
  - assign a certain literal to a memory space identifiable by a given identifier
  - e.g.,
    - int intValue = 3;
    - long longValue = 5l;
    - float floatValue = 1.2f;
    - char charValue = 'a';
    - boolean boolValue = true;
    - String stringValue = "Hello Java";
    - int[] intArray = new int[]{1,2,3};
    - Circle circleValue = new Circle(1,2,3);

```java
public class P2 {
 public static void main(String[] args) {
   // express and print out some information with
   // data type, identifier, assignment operator, literal
   int semester = 5;
   System.out.println(semester);
 }
}
```

Expression

- Type Coercion
  - the automatic or implicit conversion of values from one data type to another

- Dat Type
  - Primitive Type

1bit	• boolean
2bytes	• char
1byte	• byte
2bytes	• short
4bytes	• int
	• long
4bytes	• float
8bytes	• double

a cup for int

another cup for long

Expression

- Type Coercion
  - the automatic or implicit conversion of values from one data type to another

- Dat Type
  - Primitive Type

1bit	• boolean
2bytes	• char
1byte	• byte
2bytes	• short
4bytes	• int
8bytes	• long
4bytes	• float
8bytes	• double

```
public class P3 {
        public static void main(String[] args) {
                // Practice type coercion

        }
}
```

Operator

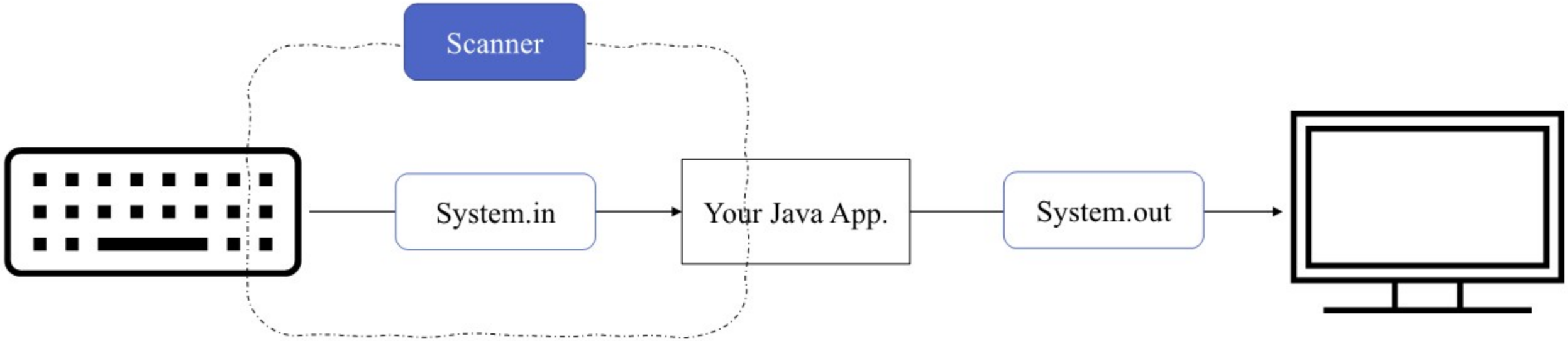
- Operator
  - manipulate a certain value or operand
  - Arithmetic operators
    - +, -, *, /, %
  - Assignment operators
    - =, +=, -=, *=, ...
  - Incremental operators
    - ++, --
  - Conditional operators
    - >, <, >=, <=, ==, !=
  - Logical operators
    - &&, ||, !

```java
public class P5 {
 @SuppressWarnings("unused")
 public static void main(String[] args) {
  float attendance = 10f;
  float midterm = 30f;
  float assignment = 30f;
  float finalEx = 30f;

  float sum = 0f;
  // fill out
  System.out.println(sum);

  boolean isLargerThan81 = false;
  // fill out
  System.out.println(isLargerThan81);

  boolean isB = false;
  // fill out
  System.out.println(isB);
 }
}
```

Input

- System.out
  - standard output to a console
- System.in
  - standard input from keyboard
  - return key inputs as bytes
- Scanner built-in class
  - let System.in read keyboard inputs

```java
public class P4 {
  public static void main(String[] args) {
  // instantiate a scanner
  java.util.Scanner scanner = new java.util.Scanner(System.in);
  // get a string
  String input = scanner.next();
  // print out the string
  System.out.println(input);
  // have to be closed
  scanner.close();
 }
}
```

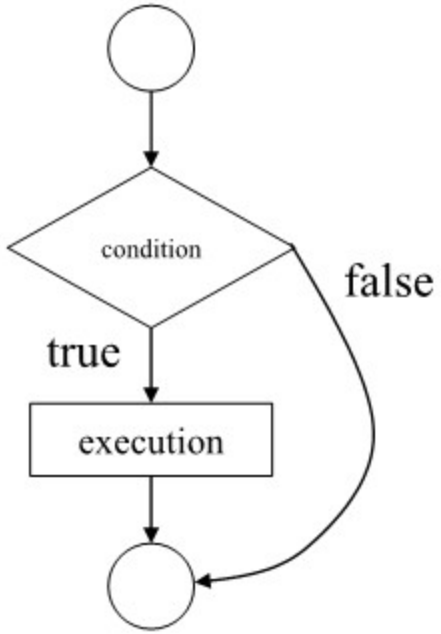


Branch

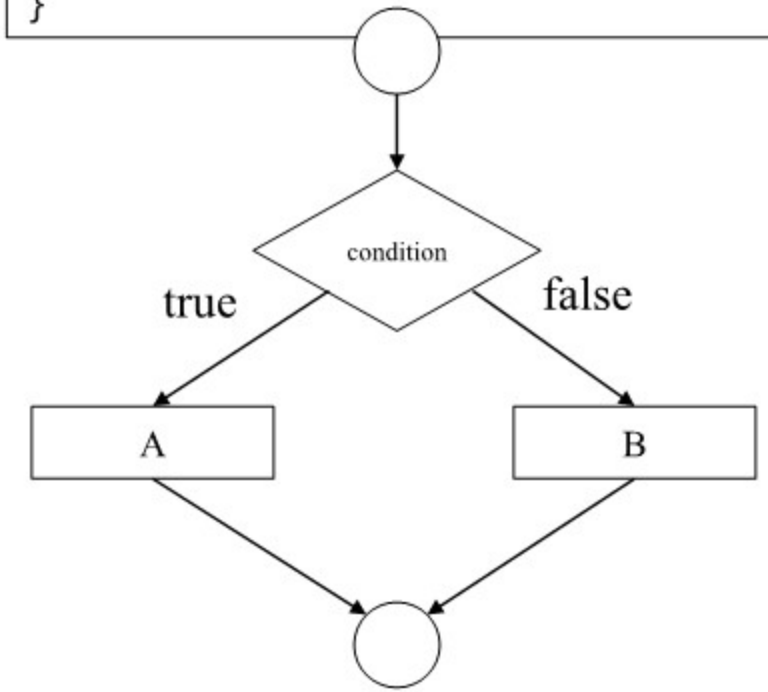
- Branch
  - an instruction that tells a computer to begin executing different instructions rather than simply executing the instructions in order

- if statement

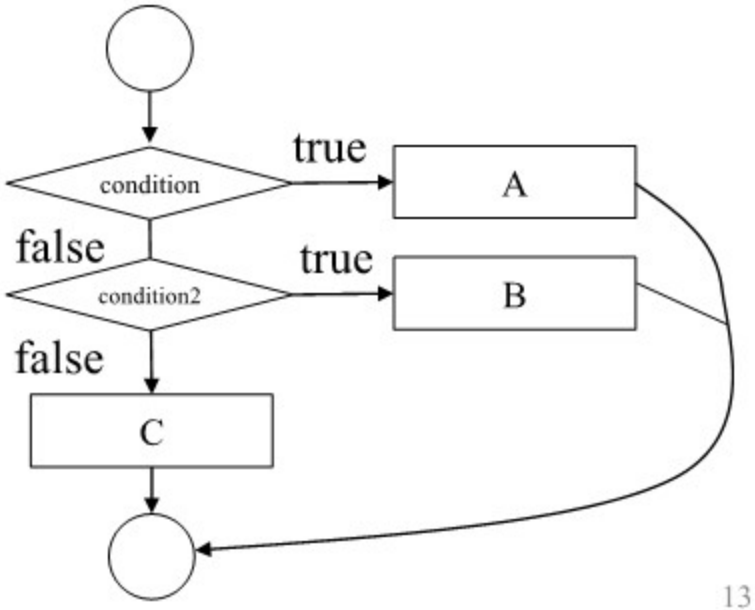
```
if(condition) {
  // do something if condition is true
}
```




```
if(condition) {
  // A: do something if condition is
true
}else {
  // B: do something if condition is
false
}
```




```
if(condition) {
 // A: do something if condition is
true
}else if(condition2) {
 // B: do something if condition2 is
true
}else {
 // C: do something if both conditions
are false
}
```



Branch

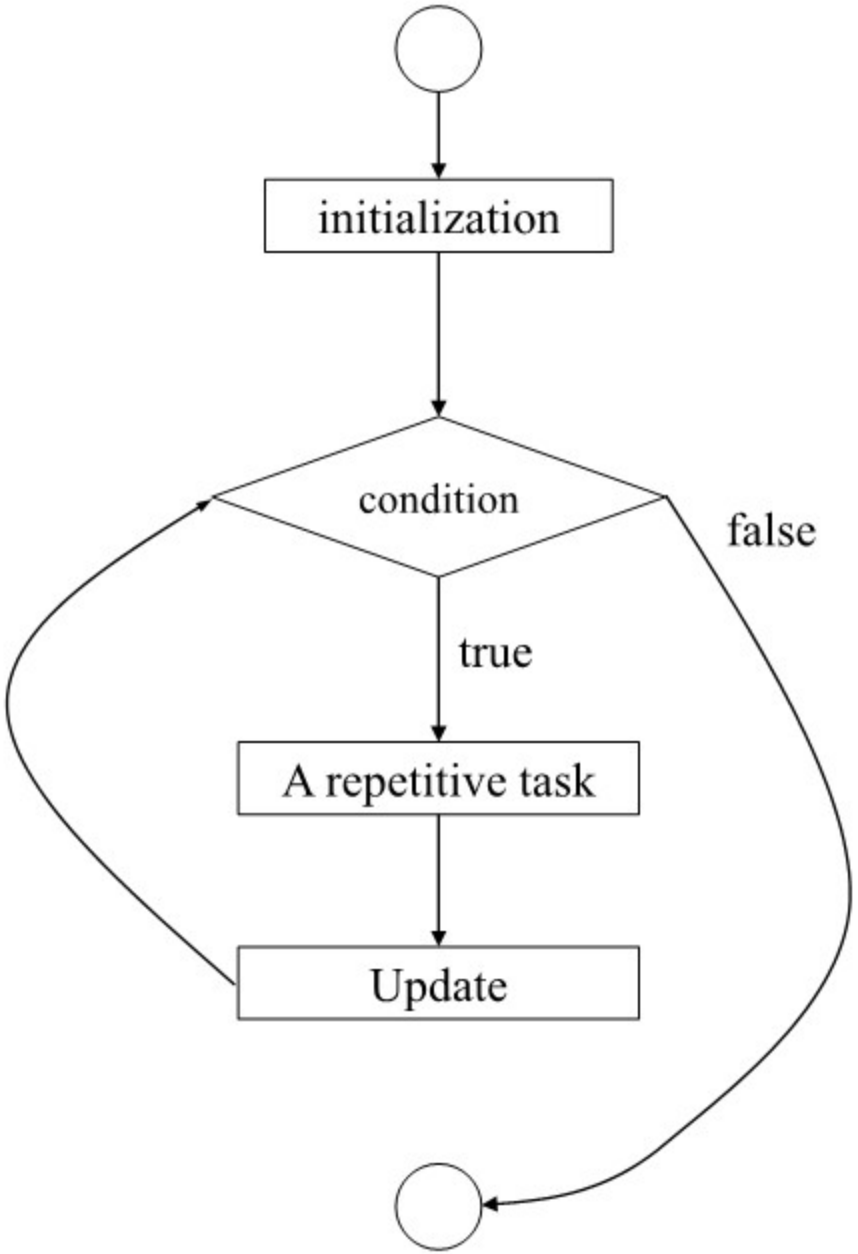
- Branch practice

```
public class P6 {
 public static void main(String[] args) {
  Scanner scan = new Scanner(System.in);

  int point = scan.nextInt();

  // Print out A, B, or F based on your point
  if(point >= 90)
   System.out.println("A");

  scan.close();
 }
}
```

Loop

- Loop statement: For

```
for( initialization ; condition ; update ) {
 // A repetitive task
}
```



Loop

- Loop statement: For
  - Practice

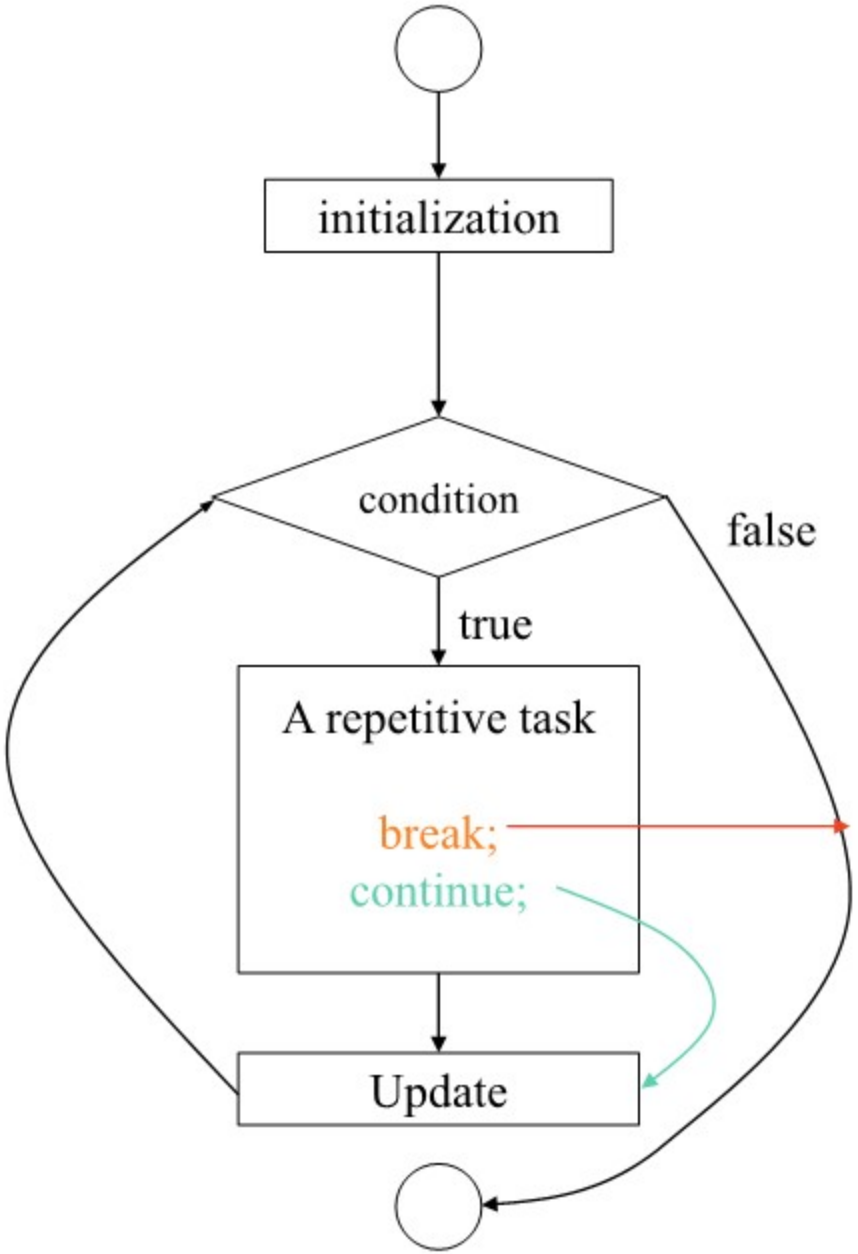
```
public class P7 {
 public static void main(String[] args) {

  int sum = 0;

  for (int i = 0; i < 5; i++) {
   sum += i;
  }

  System.out.println(sum);
 }
}
```

Loop

- Loop statement: For
  - with continue; and break;

  - break; forces to terminate loop
  - continue; forces to restart loop

```
for( initialization ; condition ; update ) {
 // A repetitive task
}
```



Loop

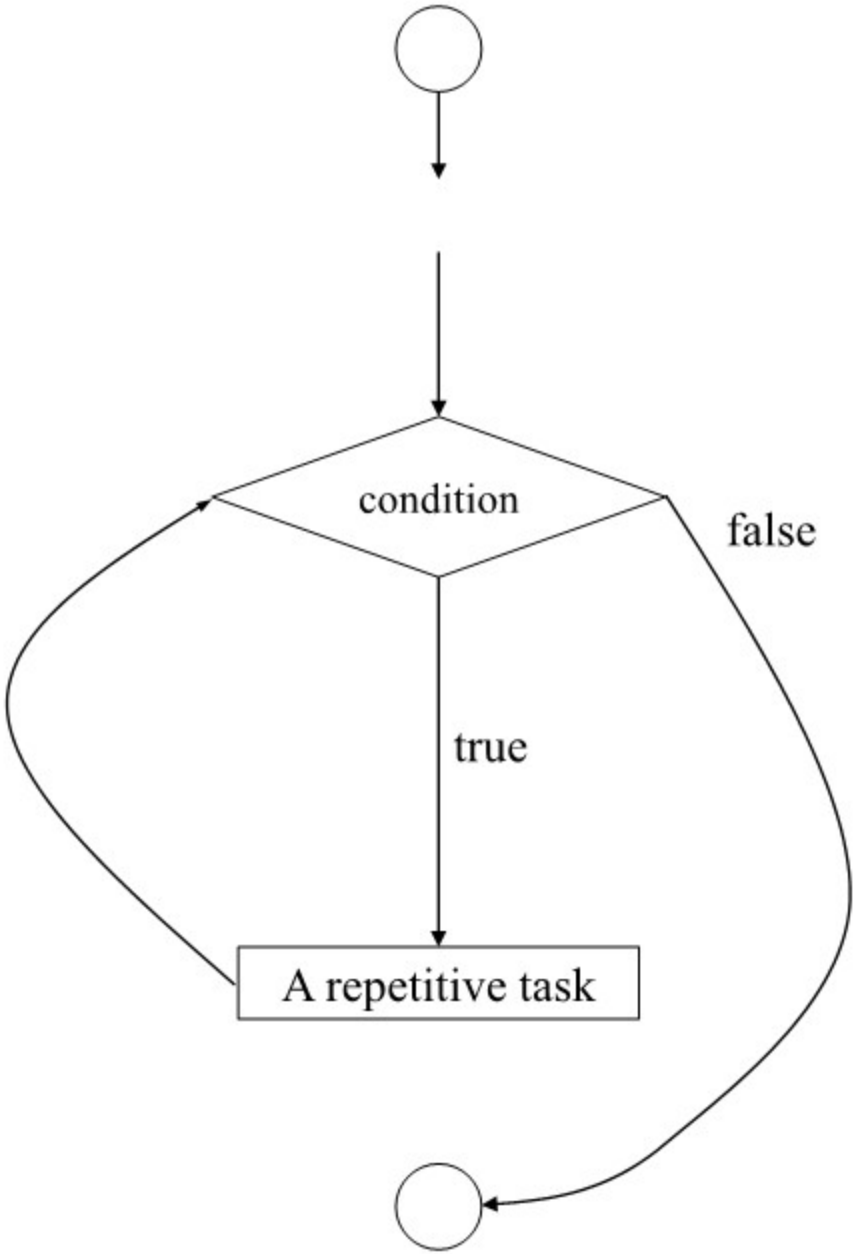
- Loop statement: For
  - Practice
    - sum positive even numbers without using condition in for statement

```
public class P7 {
 public static void main(String[] args) {

  int sum = 0;

  for (int i = 0; true; i++) {
   sum += I;
  }

  System.out.println(sum);
 }
}
```

Loop

- Loop statement: While

```
while( condition ) {
 // A repetitive task
}
```



Loop

- Loop statement: For
  - Practice
    - convert for statement to while

```
public class P7 {
 public static void main(String[] args) {

  int sum = 0;

  for (int i = 0; i < 5; i++) {
   sum += i;
  }

  System.out.println(sum);
 }
}
```

Array

- Array
  - A data structure of homogeneous values accessed by index
  - e.g.,
    - int i[]
    - long l[]
    - float f[]
    - String s[]
    - Object o[]

4	13	9	51	2	79	8	9
1st	2nd	3rd	4th	5th	6th	7th	8th
i[0]	i[1]	i[2]	i[3]	i[4]	i[5]	i[6]	i[7]

Array

- Array
  - Initialization
    - int i[] = new int[n]; // where n is the number of elements
  - Set
    - int[3] = 51;
  - Get
    - int fifthValue = int[4];
  - Iteration
    - using for statement

4	13	9	51	2	79	8	9
1st	2nd	3rd	4th	5th	6th	7th	8th
i[0]	i[1]	i[2]	i[3]	i[4]	i[5]	i[6]	i[7]

Array

- Array
  - Practice

```
public class P8 {
 public static void main(String[] args) {

  int intArr[] = new int[10];

  for(int i = 0 ; i < 10 ; i++) {
   intArr[i] = i+1;
  }

  int sum = 0;

  for(int i = 0 ; i < 10 ; i++) {
   sum += intArr[i];
  }

  System.out.println(sum);
 }
}
```

Method

- Method
  - a collection of statements that perform some specific task and return the result to the caller (if exists)

```
RETURN_TYPE METHOD_NAME(PARAM_TYPE1 PARAM_VAR1, PARAM_TYPE2 PARAM_VAR2, ...) {
  // TASK
  return A VALUE of RETURN_TYPE;
}

public static void main(String[] args) {
 RETURN_TYPE RETURN_VALUE = METHOD_NAME(PARAM_VAL1, PARAM_VAL2, ...);
}
}
```

argument values or actual parameters

Method

- Method
  - a collection of statements that perform some specific task and return the result to the caller (if exists)
  - Practice

```
public class P9 {

 public static int sum(int x, int y) {
  return x + y;
 }

 public static void main(String[] args) {
  System.out.println(sum(3,5));
 }
}
```

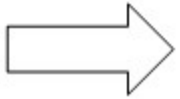
Exception Handling

- Exception
  - Error can yield malfunction in run-time
  - e.g.,
    - Divide by zero
    - Array index out of bound
    - File not found
    - Number format exception
  - Java
    - Exception -> JVM recognizes it -> JVM notifies it to the App.
      - if no exception handling -> terminate the App.

Exception Handling

- Exception
  - try-catch
  - throw

```
public class P10 {

 public static void main(String[] args) {
  Scanner scan = new Scanner(System.in);

  int point = scan.nextInt();

  // A, B, F를 출력하시오
  if (point >= 90)
   System.out.println("A");

  scan.close();
 }
}
```



Class

- Object-oriented programming
  - a computer programming paradigm that organizes software design around 'objects' rather than functions and logic

- We abstract everything as Class
  - with data abstraction
  - with procedure abstraction

- We achieve a specific goal by using objects (i.e., instances) of Classes
  - Class: design, blueprints
  - Object: instance

Class

- Object-oriented programming
  - a computer programming paradigm that organizes software design around 'objects' rather than functions and logic

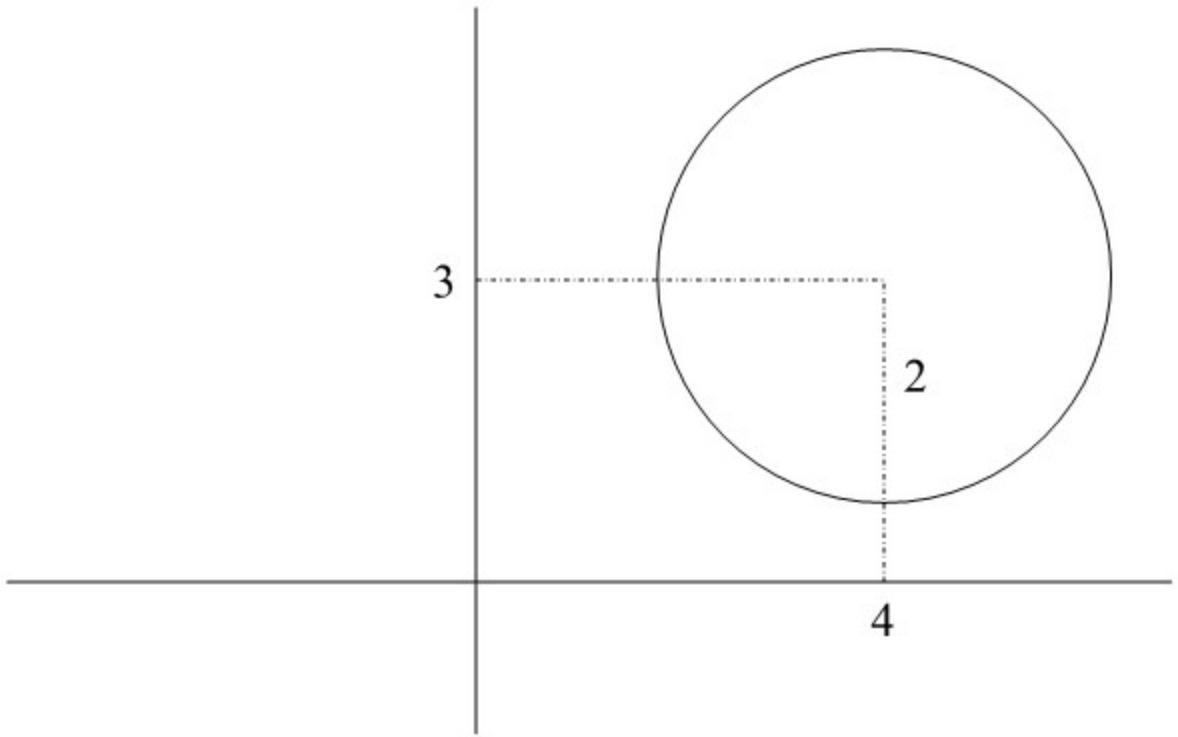
- We abstract everything as Class
  - with data abstraction
  - with procedure abstraction

- We achieve a specific goal by using objects (i.e., instances) of Classes
  - Class: design, blueprints
  - Object: instance

```
class CLASS_NAME {
    // DATA ABSTRACTION

    // PROCEDURE ABSTRACTION
}
```

Class

- Class
  - Practice: Circle
    - Circle is an object that has its x-axis y-axis values for its center and radius



Class

- Class
  - Practice: Circle
    - Circle is an object that has its x-axis y-axis values for its center and radius
    - An instance of Circle has one functionality to show its area

```java
public class Circle {
 // DATA ABSTRACTION
 int x;
 int y;
 int r;
 // PROCEDURE ABSTRACTION
 float getArea() {
  return 3.14f * r * r;
 }
}
```

```java
public class P11 {

 public static void main(String[] args) {
  Circle c1 = new Circle();
  c1.x = 4;
  c1.y = 3;
  c1.r = 2;
  System.out.println(c1.getArea());
  Circle c2 = new Circle();
  c2.x = 2;
  c2.y = 3;
  c2.r = 4;
  System.out.println(c2.getArea());
 }
}
```

Class: Constructor

- Constructor
  - Constructor allows initializing an object when it is created

```java
public class Circle {
 // DATA ABSTRACTION
 int x;
 int y;
 int r;
 // PROCEDURE ABSTRACTION
 public Circle(int x, int y, int r) {
  this.x = x;
  this.y = y;
  this.r = r;
 }

 float getArea() {
  return 3.14f * r * r;
 }
}
```

```java
public class P12 {
 public static void main(String[] args) {
  Circle c1 = new Circle(4,3,2);
  System.out.println(c1.getArea());

  Circle c2 = new Circle(2,3,4);
  System.out.println(c2.getArea());
 }
}
```

Class: Getters/Setters

- Class
  - Direct access (GET/SET) to a member variable is not recommended
  - public <- one of access modifier

```java
public class Circle {
 // DATA ABSTRACTION
 int x;
 int y;
 int r;
 // PROCEDURE ABSTRACTION
 public Circle(int x, int y, int r) {
  this.x = x;
  this.y = y;
  this.r = r;
 }

 float getArea() {
  return 3.14f * r * r;
 }
}
```

```java
public class P11 {

 public static void main(String[] args) {
  Circle c1 = new Circle();
  c1.x = 4;
  c1.y = 3;
  c1.r = 2;
  System.out.println(c1.getArea());
  Circle c2 = new Circle();
  c2.x = 2;
  c2.y = 3;
  c2.r = 4;
  System.out.println(c2.getArea());
 }
}
```

Class: Getters/Setters

- Class
  - Direct access (GET/SET) to a member variable is not recommended
  - public <- one of access modifier

Relation	private	default	protected	public
same package	x	o	o	o
inherited class	x	x	o	o
different package	x	x	x	o

Class: Getters/Setters

- Class
  - Direct access (GET/SET) to a member variable is not recommended
  - Recommended to provide methods if a member variable is needed to be accessed
  - trivial
    - GETTERS / SETTERS

```java
public class Circle {
  // DATA ABSTRACTION
  private int x;
  private int y;
  private int r;

  // PROCEDURE ABSTRACTION
  public int getX() { return x; }
  public void setX(int x) { this.x = x; }
  public int getY() { return y; }
  public void setY(int y) { this.y = y; }
  public int getR() { return r; }
  public void setR(int r) { this.r = r; }

  ...
}
```

```java
public class P13 {

 public static void main(String[] args) {
  Circle c1 = new Circle();
  c1.setX(4);
  c1.setY(3);
  c1.setR(2);
  System.out.println(c1.getArea());
 }
}
```

Class: Inheritance

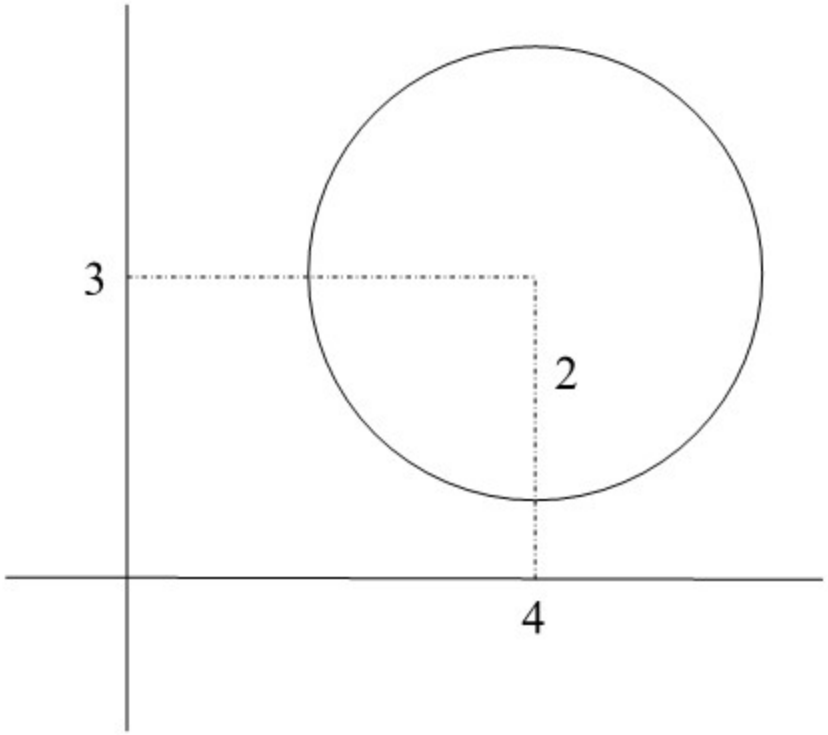
- Inheritance
  - a mechanism wherein a new class is derived from an existing class
- General vs. Specific
- The number of information
- isA

```
public class Person {
  ...
}
public class Student extends Person {
  ...
}
```

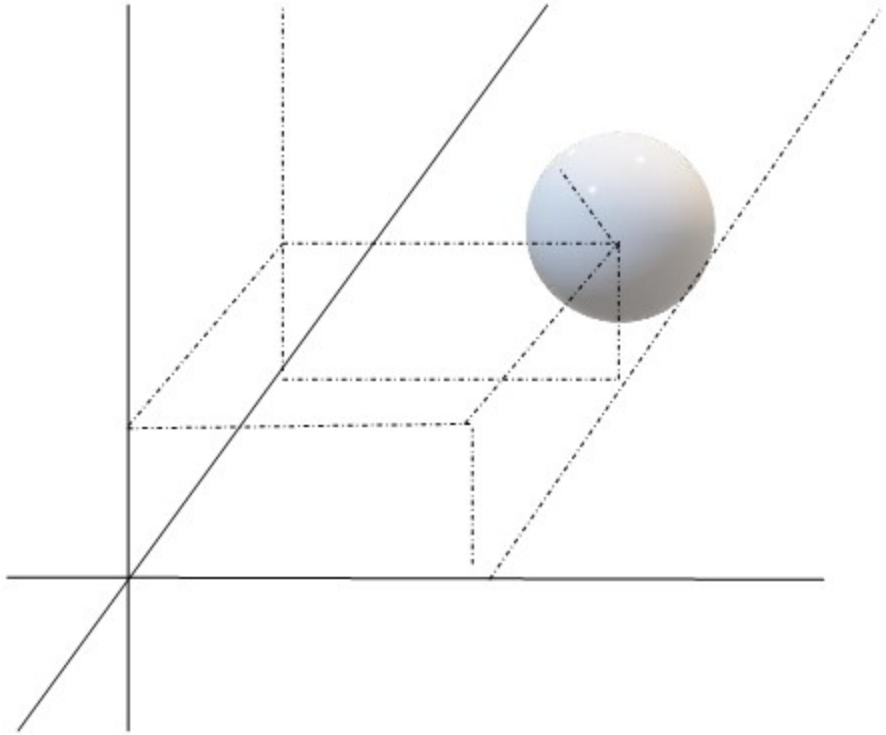
Class: Inheritance

- Practice

Parent class



Child class



Class: Inheritance

- Practice

```
public class Sphere extends Circle {
 private int z;

 public Sphere(int x, int y, int z, int r) {
  super(x, y, r);
  this.z = z;
 }

 public int getZ() {
  return z;
 }

 public void setZ(int z) {
  this.z = z;
 }

 public float getVolume() {
  return getR() * getR() * getR() * 4 / 3f * 3.14f;
 }
}
```

```
public class P14 {
 public static void main(String[] args) {
  Sphere s = new Sphere(1,2,3,4);
  System.out.println(s.getVolume());
 }
}
```

Class: Upcasting / Downcasting

- Downcasting
  - A child class instance can be called a parent class instance
  - e.g., A student is a person
- Upcasting
  - A parent class instance cannot be called a child class instance
  - e.g., it is not 100% true if we say a person is a student

- instanceof operator
  - return boolean value

```
public class P15 {
 @SuppressWarnings("unused")
 public static void main(String[] args) {
  Circle c = new Sphere(1,2,3,4);

  Sphere s = (Sphere) new Circle(3,4,5);
 }
}
```

Class: Overriding

- Overriding
  - A child class inherits methods of ancestors
  - Sometimes the child class wants to re-define a specific inherited method

- For example
  - Every class implicitly inherits java.lang.Object
  - Object has toString method
  - System.out.println(Object obj) uses Object.toString()
  - needed to re-define toString()

```
@Override
public String toString() {
 return "(" + x + "," + y + ")-" + r;
}
```

```
public class P16 {
 @SuppressWarnings("unused")
 public static void main(String[] args) {
  Circle c = new Circle(1,2,3);
  System.out.println(c);
 }
}
```

Class: Generics and Polymorphism

- Generics
  - a facility of generic programming that were added to J2SE 5.0
  - enables to specify a set of related methods working for a set of related types (even custom class) with a single method declaration

- Assume a very simple Printer class that
  - has just one data
  - has just one method to print it out

- Can we design a printer for Circle instance (will be made 10 years later)

Class: Generics and Polymorphism

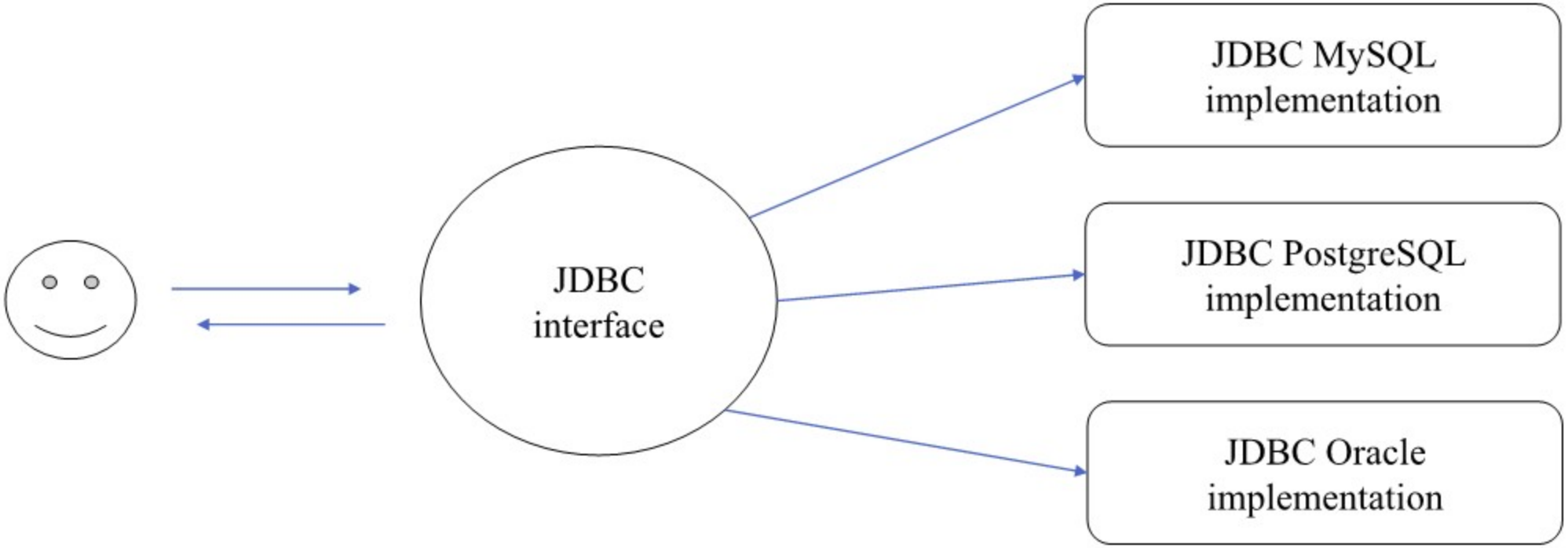
- Practice
  - GenericPrinter

```
public class GenericPrinter<E> {

 private E data;

 public GenericPrinter(E data) {
  this.data = data;
 }

 public void print() {
  System.out.println(data);
 }

}
```

```
public class P17 {
 public static void main(String[] args) {
  GenericPrinter<Circle> p =
         new GenericPrinter<Circle>(new Circle(3,4,5));
  p.print();
 }
}
```

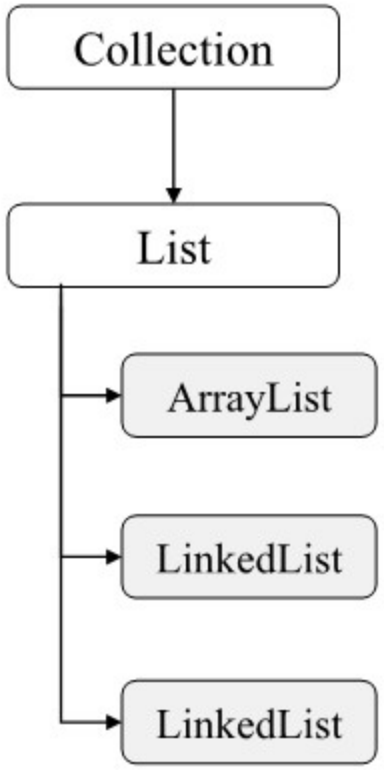
Interface

- Interface
  - Can we design common functionalities that do not depend on their implementations
  - e.g., JDBC



Interface

- Interface
  - java.util.List interface
    - An ordered collection allowing redundancy of its elements
    - Can be manipulated based on its index



Redundancy	Ordered	Thread-Safe
O	O	?

Redundancy	Ordered	Thread-Safe
O	O	X

Redundancy	Ordered	Thread-Safe
O	O	X

Redundancy	Ordered	Thread-Safe
O	O	O

Interface

- Interface
  - java.util.List interface
    - An ordered collection allowing redundancy of its elements
    - Can be manipulated based on its index

Return Type	Method	Description
boolean	isEmpty()	Check empty
int	size()	Return a size of Collection
boolean	add(E e)	Add a new instance to collection
void	add(int index, E element)	Add a new instance to collection at a specific index
boolean	contains(Object o)	Check collection contains o
E	get(int index)	Retrieve instance at a specific index
int	indexOf(Object o)	Find an index where o is located at (from 0 index)
int	lastIndexOf(Object o)	Find an index where o is located at (from size-1 index)
...	...	...

Interface

- Interface
  - Practice: Array-based List