

Algoritma dan Struktur Data

STACK

Amil Ahmad Ilham

Review Array

- Contoh menyisip item pada index 1

index	Shift down item at n-1	Shift down item at n-2	Shift down item at i	Now safe to replace item at position 1	Array after insertion is finished
0	D1	D1	D1	D1	D1
1	D2	D2	D2	D2	D5
2	D3	D3	D3	D2	D2
3	D4	D4	D3	D3	D3
4		D4	D4	D4	D4

```
//Shift items down by one position
for (int i = logicalSize; i > targetIndex ; i-- )
{
    array[i] = array[i-1];
}
//Add new item and increment logical size
array[targetIndex] = newItem;
logicalSize++;
```

Review Array

- Contoh menghapus item pada index 1

index	Shift up item at i+1	Shift up item at i+2	Shift up item at n-1	Array after removal is finished
0	D1	D1	D1	D1
1	D2	D3	D3	D3
2	D3	D3	D4	D4
3	D4	D4	D4	D5
4	D5	D5	D5	

```
//Shift items up by one position
for (int i = targetIndex; i < logicalSize - 1; i++)
{
    array[i] = array[i+1];
}
//Decrement logical size
logicalSize--;
```

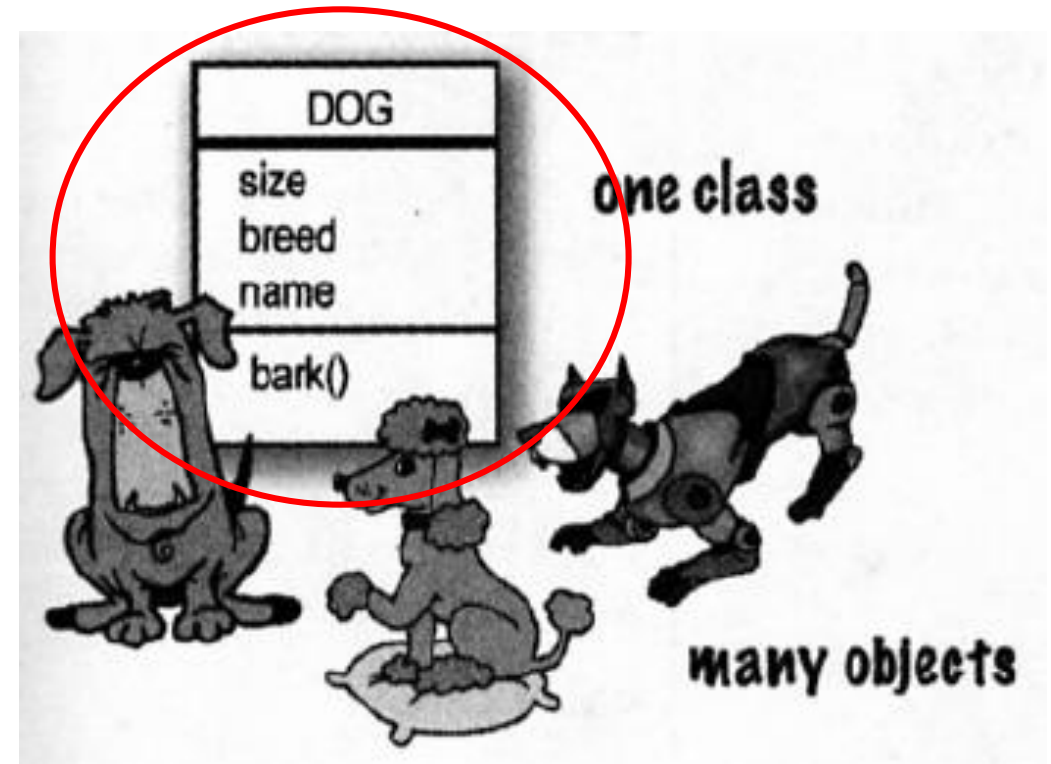
Review Java

- Java adalah bahasa pemrograman berorientasi object
- Program java nampak seperti sekumpulan object yang bekerja bersama-sama menyelesaikan suatu tugas yang diberikan.

Review Java

- **Class**

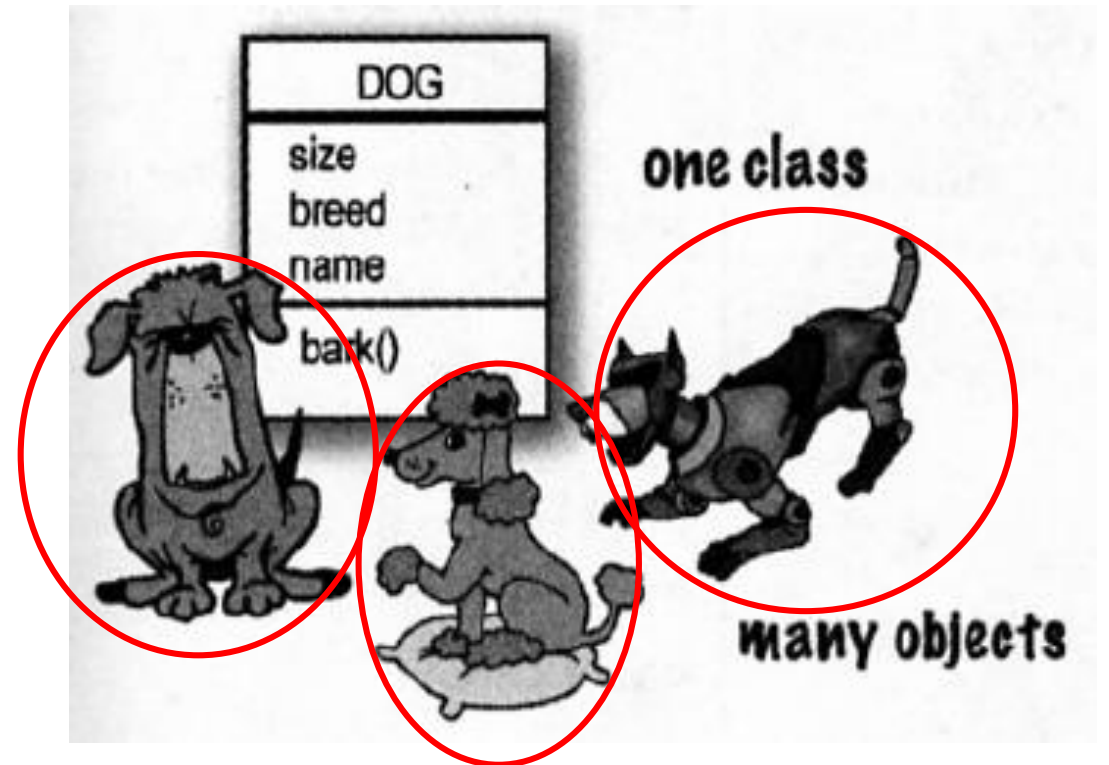
- Prototype atau template yang mendefinisikan suatu object
- Menggambarkan apa yang object miliki dan apa yang object dapat kerjakan
- Contoh: Class Dog
 - Dog memiliki: size, breed, name
 - Dog dapat: bark()
- Apa yang object miliki disebut dengan **instance variables**
- Apa yang object dapat kerjakan disebut dengan **methods**



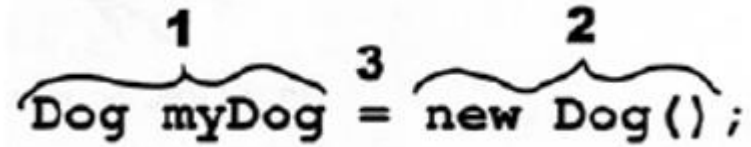
Review Java

- **Object**

- Instance dari sebuah class
- Satu class, banyak object
- Contoh: Class Dog
Object: dog1, dog2, dog3



Membuat object



The diagram shows the Java code `Dog myDog = new Dog();` with three numbered annotations. A bracket labeled '1' is above `Dog myDog`. A bracket labeled '2' is above `new Dog()`. The equals sign `=` is labeled with '3'.

```
Dog myDog = new Dog();
```

1 Declare a reference variable

2 Create an object

3 Link the object and the reference

Implementasi OOP

- Buat class

```
// Class Dog

public class Dog
{
    //instance variables
    public int size;
    public String breed;
    public String name;

    //method
    public void bark()
    {
        System.out.println("This dog has\nbreed: "+breed+ "\n" +
            "size: "+size+ "\n" +
            "name: "+name);
    }
}
```


Implementasi OOP

- Buat class test yang berisi method “main” dan object

```
/* Class DogTest
 * Berisi method "main" yang menunjukkan bahwa class ini bisa di run
 * Berisi object dog
 */

public class DogTest
{
    public static void main(String[] args)
    {
        //membuat object myDog
        Dog myDog = new Dog();

        //mengakses variabel di class Dog
        myDog.size = 40;
        myDog.breed = "bulldog";
        myDog.name = "Wangwang";

        //mengakses method di class Dog
        myDog.bark();

        //Tugas#1: buat object dog yang lain
    }
}
```

Implementasi OOP

- Contoh lain class test

```
// Class DogTest

public class DogTestArray
{
    public static void main(String[] args)
    {
        //membuat array object dog
        Dog[] dog = new Dog[3];

        //membuat object dog
        dog[0] = new Dog();
        dog[0].size = 40;
        dog[0].breed = "bulldog";
        dog[0].name = "Wangwang";

        dog[0].bark();

        //Tugas#2: buat object dog yang lain
    }
}
```

Struktur Data

Struktur Data

Good software

- Requirements for a good software:
 - Clean Design
 - Easy maintenance
 - Reliable
 - Easy to use
 - Fast algorithms
- **Efficient data structures**
- **Efficient algorithms**

Why?

- Computers take on more and more complex tasks
 - Imagine: index of 8 billion pages ! (Google)
- Software implementation and maintenance is difficult.
- Clean conceptual framework allows for more efficient and more correct code

Collections

- Collection is a group of items that we wish to treat as a conceptual unit.
- Categories of Collections:



Figure 1.1 A linear collection

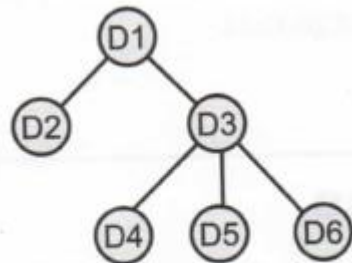


Figure 1.2 A hierarchical collection

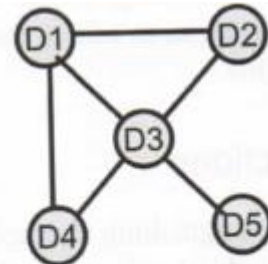


Figure 1.3 A graph collection



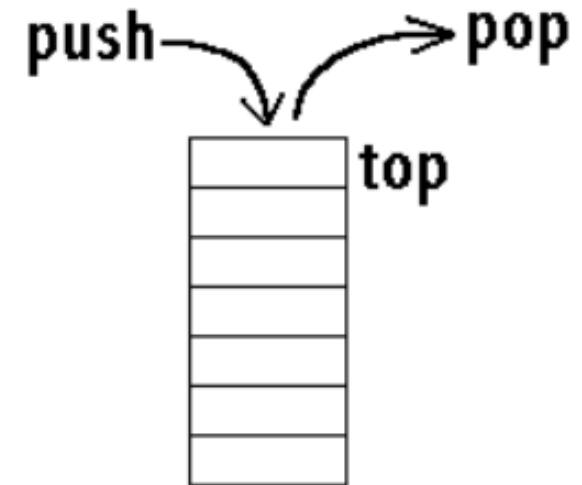
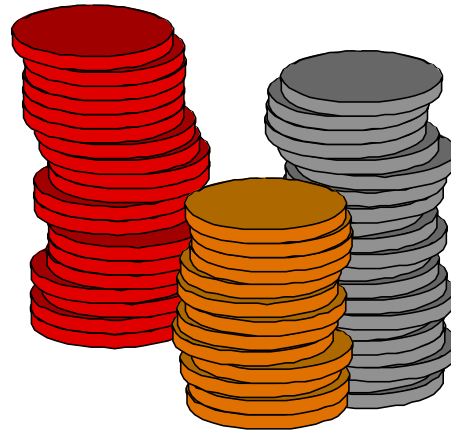
Figure 1.4 An unordered collection

Linear Collections

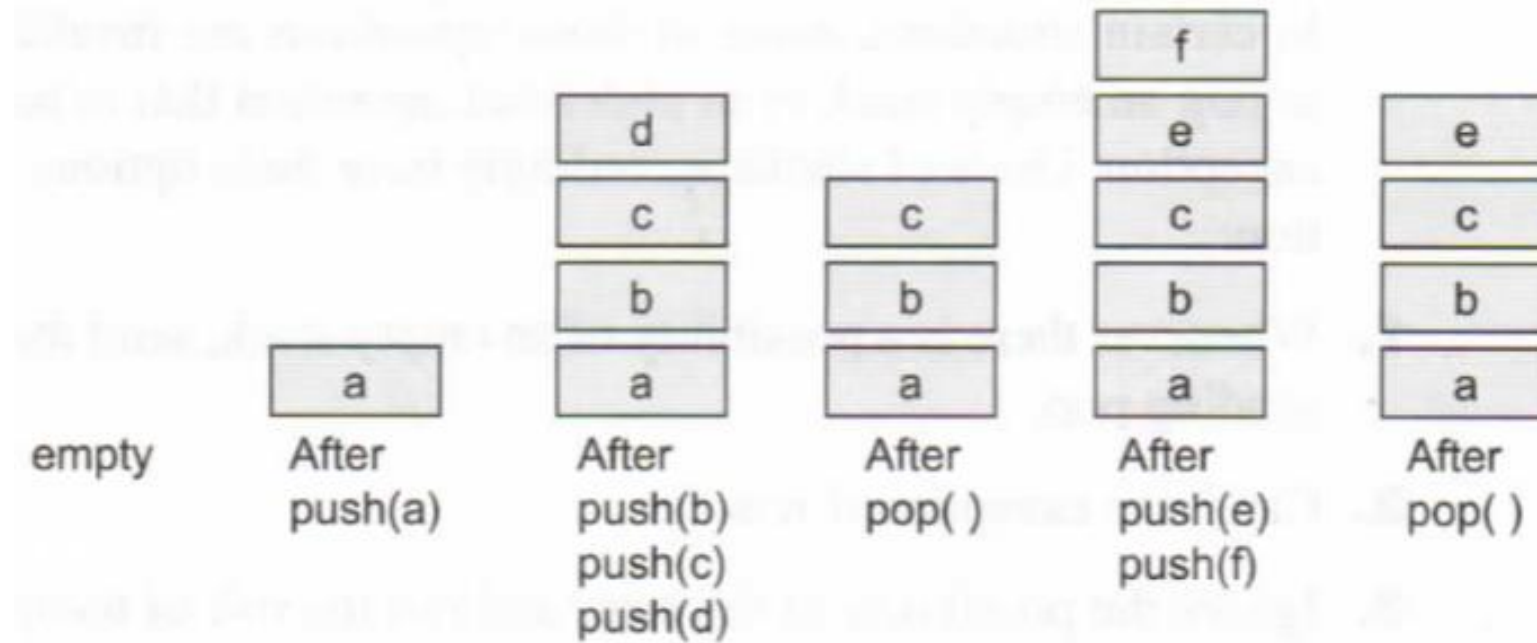
- Commonly used linear collections:
 - String
 - Lists
 - **Stacks**
 - **Queues**

Stacks

- A stack is a LIFO (LAST IN FIRST OUT) sequence.
 - Addition and removal takes place **only** at one end, called the top.
- Examples:
 - Stacks of plates
 - Trains
 - Vending Machines
 - Expression Evaluation
 - Navigating a maze
 - Map coloring



Lifetime of a Stack



The Effects of Stack Operations

Operation	State of the Stack After the Operation	Value Returned	Comment
			Initially, the stack is empty.
push(a)	a		The stack contains the single item a.
push(b)	a b		b is the top item on the stack.
push(c)	a b c		c is the top item.
isEmpty()	a b c	false	The stack is not empty.
size()	a b c	3	The stack contains three items.
peek()	a b c	c	Return the top item on the stack without removing it.
pop()	a b	c	Remove the top item from the stack and return it. b is now the top item.
pop()	a	b	Remove and return b.
pop()		a	Remove and return a.
isEmpty()		true	The stack is empty.
peek()		exception	Peeking at an empty stack throws an exception.
pop()		exception	Popping an empty stack throws an exception.
push(d)	d		d is the top item.

The Interface for the Stack Prototype (StackPT)

void

push(Object item)

Pushes an item onto the top of this stack. Throws an exception if the item is null or the stack is full.

Object

pop()

Returns the item at the top of this stack and removes it from the stack. Throws an exception if the stack is empty.

Supporting Methods

Object

peek()

Returns the item at the top of this stack without removing it from the stack. Throws an exception if the stack is empty.

boolean

isEmpty()

Returns true if this stack contains no items.

boolean

isFull()

Returns true if this stack is full and can accept no more items.

int

size()

Returns the number of items in this stack.

Implementation of the Stack Prototype Using Array

1. Stack.java
2. ArrayStack.java
3. StackEmptyException.java
4. StackFullException.java
5. StackTest.java

Stack.java

- adalah interface yang berisi daftar method yang digunakan untuk implementasi stack.

```
//Interface Stack

public interface Stack {

    //return number of elements in the stack
    public int size();

    //Return whether the stack is full
    public boolean isFull();

    //Return whether the stack is empty.
    public boolean isEmpty();

    //return top element in the stack.
    public Object peek() throws StackEmptyException;

    //Insert an element at the top of the stack.
    public void push (Object item) throws StackFullException;

    //Remove the top element from the stack.
    public Object pop() throws StackEmptyException;

}
```

ArrayStack.java

- Implementasi interface Stack menggunakan Array

```
//Stack implementation using Array

public class ArrayStack implements Stack {

    //Default length of the array used to implement the stack.
    public static final int CAPACITY = 1000;

    private int capacity; //max stack size
    private Object S[]; //Array used to implement the stack.
    private int top = -1; //Index of the top element of the stack.

    //Default constructor
    public ArrayStack() {
        this(CAPACITY);
    }

    //Alternatif constructor
    public ArrayStack(int cap) {
        capacity = cap;
        S = new Object[capacity];
    }

    //return number of elements in the stack
    public int size() {
        return (top + 1);
    }

    //Return whether the stack is full
    public boolean isFull() {
        return (size() == capacity);
    }
}
```

ArrayStack.java

- Implementasi interface Stack menggunakan Array

```
//Return whether the stack is empty
public boolean isEmpty() {
    return (top < 0);
}

//Return top element in the stack
public Object peek() throws StackEmptyException {
    if (isEmpty())
        throw new StackEmptyException("Stack is empty.");
    return S[top];
}

//Insert an element at the top of the stack
public void push(Object obj) throws StackFullException {
    if (isFull())
        throw new StackFullException("Stack is full.");
    top = top + 1;
    S[top] = obj;
}

//Remove the top element from the stack
public Object pop() throws StackEmptyException {
    Object elem;
    if (isEmpty())
        throw new StackEmptyException("NO MORE POP! Stack is empty.");
    elem = S[top];
    top = top - 1;
    return elem;
}
}
```


StackEmptyException.java

- adalah class yang dipanggil jika operasi peek atau pop dilaksanakan pada saat stack sudah kosong.

```
// Empty Stack Exception
public class StackEmptyException extends RuntimeException {
    public StackEmptyException(String err){
        super(err);
    }
}
```


StackFullException.java

- adalah class yang dipanggil jika operasi push dilaksanakan pada saat stack sudah penuh.

```
// Full Stack Exception
public class StackFullException extends RuntimeException {
    public StackFullException(String err){
        super(err);
    }
}
```

StackTest.java

- Class yang berisi method 'main'
- Berisi object ArrayStack

```
//Testing the Stack
```

```
public class StackTest {  
  
    public static void main(String[] args)  
    {  
        int N=9;  
  
        //Buat object ArrayStack  
        ArrayStack myStack = new ArrayStack(N);  
  
        System.out.println("Operasi PUSH");  
  
        for (int i=0; i<N;i++ )  
        {  
            myStack.push("Piring-"+(i+1));  
            System.out.print("Item paling atas: " + myStack.peek());  
            System.out.println(" Jumlah item di Stack: " + myStack.size());  
        }  
  
        System.out.println("Operasi POP");  
        for (int i=0; i<N; i++)  
        {  
            System.out.println("Item paling atas: " + myStack.peek());  
            System.out.print("Item yang di keluarkan: " + myStack.pop());  
            System.out.print(" Sisa item di Stack: " + myStack.size()+" ");  
            //System.out.println("Item paling atas:: " + myStack.peek());  
        }  
    }  
}
```

Output StackTest.java

```
D:\01tutorial\struktur_data\tut2>javac *.java

D:\01tutorial\struktur_data\tut2>java StackTest
Operasi PUSH
Item paling atas: Piring-1 Jumlah item di Stack: 1
Item paling atas: Piring-2 Jumlah item di Stack: 2
Item paling atas: Piring-3 Jumlah item di Stack: 3
Item paling atas: Piring-4 Jumlah item di Stack: 4
Item paling atas: Piring-5 Jumlah item di Stack: 5
Item paling atas: Piring-6 Jumlah item di Stack: 6
Item paling atas: Piring-7 Jumlah item di Stack: 7
Item paling atas: Piring-8 Jumlah item di Stack: 8
Item paling atas: Piring-9 Jumlah item di Stack: 9
Operasi POP
Item paling atas: Piring-9
Item yang di dikeluarkan: Piring-9 Sisa item di Stack: 8 Item paling atas: Piring-8
Item yang di dikeluarkan: Piring-8 Sisa item di Stack: 7 Item paling atas: Piring-7
Item yang di dikeluarkan: Piring-7 Sisa item di Stack: 6 Item paling atas: Piring-6
Item yang di dikeluarkan: Piring-6 Sisa item di Stack: 5 Item paling atas: Piring-5
Item yang di dikeluarkan: Piring-5 Sisa item di Stack: 4 Item paling atas: Piring-4
Item yang di dikeluarkan: Piring-4 Sisa item di Stack: 3 Item paling atas: Piring-3
Item yang di dikeluarkan: Piring-3 Sisa item di Stack: 2 Item paling atas: Piring-2
Item yang di dikeluarkan: Piring-2 Sisa item di Stack: 1 Item paling atas: Piring-1
Item yang di dikeluarkan: Piring-1 Sisa item di Stack: 0
```

Tugas #3

- Modifikasi class StackTest.java untuk melakukan operasi Push dan Pop sebanyak 2 angka digit terakhir dari STB anda.

SIKOLA

- Tugas#1 dan Tugas#2
 - Print screen hasil running program pada Tugas#1 (slide 9), Tugas#2 (slide 10).
Paste hasil print screen tersebut di Sikola menu Forum.
- Tugas#3.
 - Print screen hasil running program pada Tugas#3 (pdf). File pdf ini bersama dengan file java (.java) dizip dalam 1 file zip atau rar kemudian diupload ke Sikola di menu Tugas. Deadline lihat di menu Tugas.