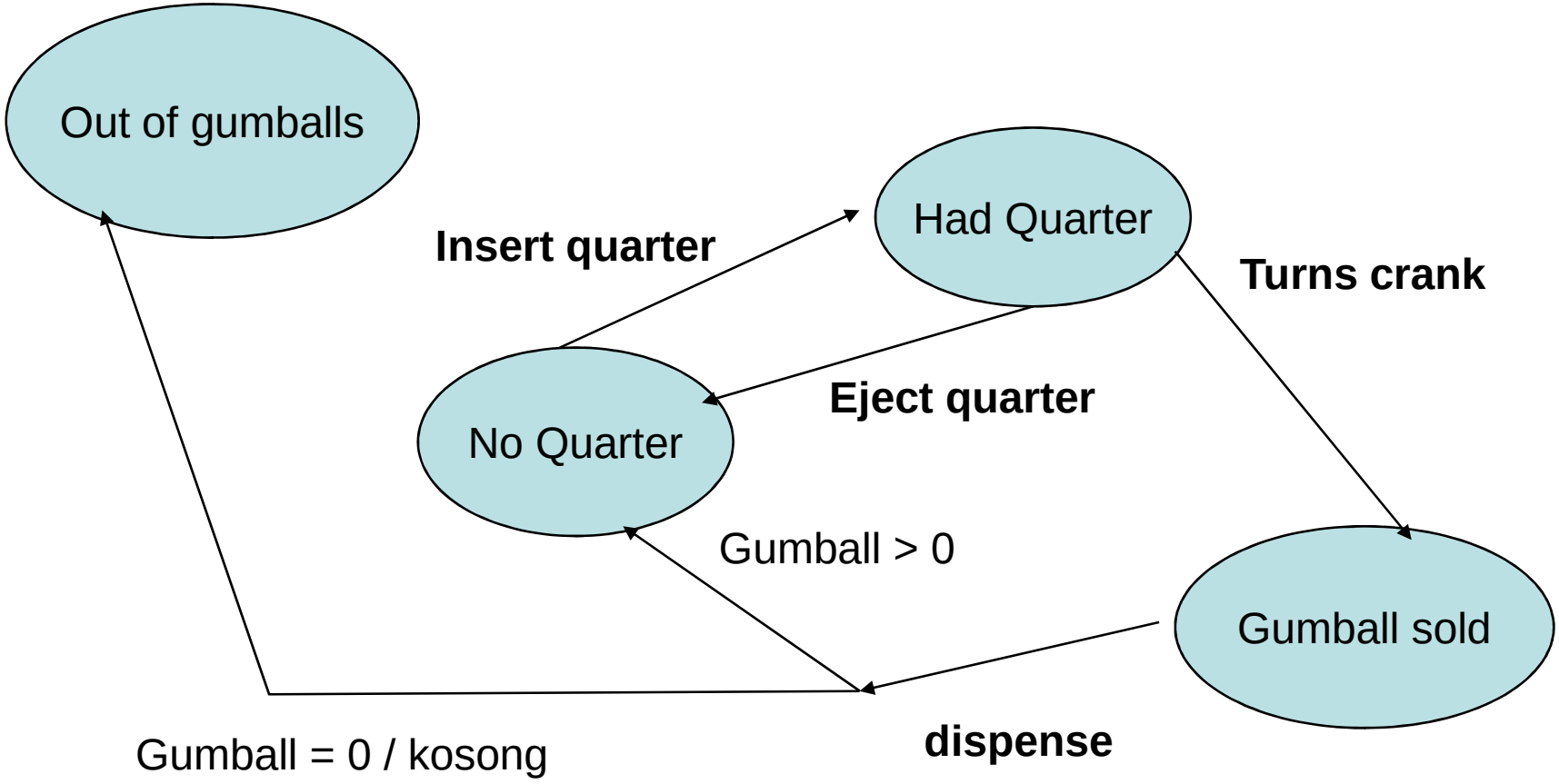
PBK 6

State Patterns

GumBall Machine

- Ada permen
- Ada mesin
- Ada tuas penarik
- Ada tempat masuk koin

Kasus Pembuatan Mesin Permen Bola (**Gumball**)



Keterangan

- Terdapat **4 state**:
  - No quarter
  - Has quarter
  - Out of gumball
  - Gumball sold out
- Ada **4 kegiatan / tindakan /aksi**:
  - Insert quarter
  - Eject quarter
  - Turn crank
  - dispense
- Semua **aksi** menyebabkan **perubahan state**, kecuali aksi **dispense**, yang “cuma” memeriksa state dan mengeluarkan gumball nya

Bagaimana implementasinya?

- Buat representasi **state**
- Buat **method** yang meng-handle semua state dan perpindahannya!
- Buat dalam class **GumballMachine**

```
public class GumballMachine{
    //konstruktor, method dan state
}
```

Implementasi State

- Dibuat dalam **KONSTANTA**
  - final static int **OUT_OF_GUMBALL** = 0;
  - final static int **NO_QUARTER** = 1;
  - final static int **HAS_QUARTER** = 2;
  - final static int **SOLD** = 3;

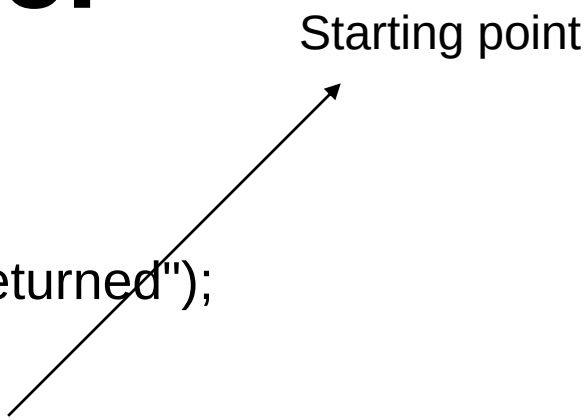
- Variabel-variabel lainnya:
  - int state = **OUT_OF_GUMBALL**; //inisialisasi awal, kosong
  - int count = 0; //jumlah permennya

insertQuarter

Starting point

```
public void insertQuarter() {
        if (state == HAS_QUARTER) {
                System.out.println("You can't insert another quarter");
        } else if (state == NO_QUARTER) {
                state = HAS_QUARTER;
                System.out.println("You inserted a quarter");
        } else if (state == OUT_OF_GUMBALL) {
                System.out.println("You can't insert a quarter, the
    machine is sold out");
        } else if (state == SOLD) {
                System.out.println("Please wait, we're already giving
    you a gumball");
        }
    }
```

ejectQuarter




```
public void ejectQuarter() {
        if (state == HAS_QUARTER) {
                System.out.println("Quarter returned");
                state = NO_QUARTER;
        } else if (state == NO_QUARTER) {
                System.out.println("You haven't inserted a quarter");
        } else if (state == SOLD) {
                System.out.println("Sorry, you already turned the
    crank");
        } else if (state == OUT_OF_GUMBALL) {
                System.out.println("You can't eject, it’s empty");
        }
    }
```

turnCrank

Starting point

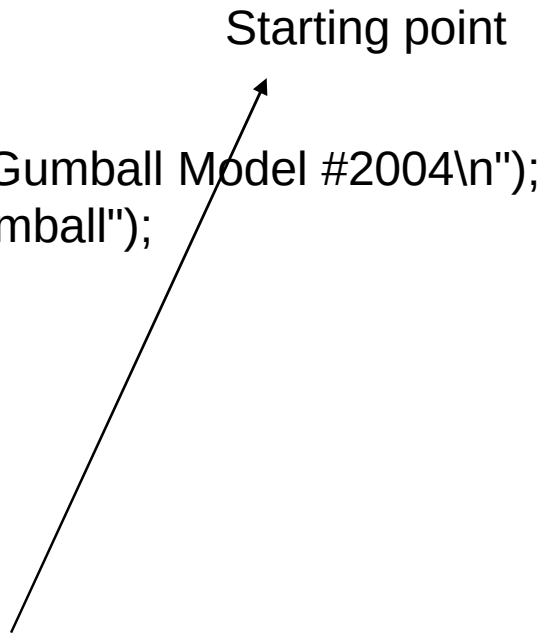
```
public void turnCrank() {
        if (state == SOLD) {
                System.out.println("Turning twice doesn't get you
    another gumball!");
        } else if (state == NO_QUARTER) {
                System.out.println("You turned but there's no quarter");
        } else if (state == OUT_OF_GUMBALL) {
                System.out.println("You turned, but there are no
    gumballs");
        } else if (state == HAS_QUARTER) {
                System.out.println("You turned...");
                state = SOLD;
                dispense();
        }
}
```

dispense

Starting point

```
public void dispense() {
        if (state == SOLD) {
                System.out.println("A gumball comes rolling out the slot");
                count = count - 1;
                if (count == 0) {
                        System.out.println("Oops, out of gumballs!");
                        state = OUT_OF_GUMBALL;
                } else {
                        state = NO_QUARTER;
                }
        } else if (state == NO_QUARTER) {
                System.out.println("You need to pay first");
        } else if (state == OUT_OF_GUMBALL) {
                System.out.println("No gumball dispensed");
        } else if (state == HAS_QUARTER) {
                System.out.println("No gumball dispensed");
        }
}
```

```
public String toString() {
        StringBuffer result = new StringBuffer();
        result.append("\nMighty Gumball, Inc.");
        result.append("\nJava-enabled Standing Gumball Model #2004\n");
        result.append("Inventory: " + count + " gumball");
        if (count != 1) {
                result.append("s");
        }
        result.append("\nMachine is ");
        if (state == OUT_OF_GUMBALL) {
                result.append("sold out");
        } else if (state == NO_QUARTER) {
                result.append("waiting for quarter");
        } else if (state == HAS_QUARTER) {
                result.append("waiting for turn of crank");
        } else if (state == SOLD) {
                result.append("delivering a gumball");
        }
        result.append("\n");
        return result.toString();
}
```



Method untuk menampilkan data mesin!

Inisialisasi konstruktor

```
public void refill(int numGumBalls) {
        this.count = numGumBalls;
        state = NO_QUARTER;
}


public GumballMachine(int count) {
        this.count = count;
        if (count > 0) {
                state = NO_QUARTER;
        }
}
```

Test

```
public class GumballMachineTestDrive {

   public static void main(String[] args) {
      GumballMachine gumballMachine = new GumballMachine(5);

      System.out.println(gumballMachine);

      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();

      System.out.println(gumballMachine);

      gumballMachine.insertQuarter();
      gumballMachine.ejectQuarter();
      gumballMachine.turnCrank();

      System.out.println(gumballMachine);

      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();
      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();
      gumballMachine.ejectQuarter();

      System.out.println(gumballMachine);

      gumballMachine.insertQuarter();
      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();
      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();
      gumballMachine.insertQuarter();
      gumballMachine.turnCrank();

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

```
E:\Documents\Dosen\PBK\program\bab6\gumball>java GumballMachineTestDrive

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
Quarter returned
You turned but there's no quarter

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
You haven't inserted a quarter

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter

You inserted a quarter
You can't insert another quarter
You turned...
A gumball comes rolling out the slot
You inserted a quarter
You turned...
A gumball comes rolling out the slot
Oops, out of gumballs!
You can't insert a quarter, the machine is sold out
You turned, but there are no gumballs

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 0 gumballs
Machine is sold out
```

Ternyata ada **perubahan** requirements

Messy state

```
final static int SOLD_OUT = 0;
final static int NO_QUARTER = 1;
final static int HAS_QUARTER = 2;
final static int SOLD = 3;


public void insertQuarter() {
    // insert quarter code here
}

public void ejectQuarter() {
    // eject quarter code here
}

public void turnCrank() {
    // turn crank code here
}

public void dispense() {
    // dispense code here
}
```

First, you'd have to add a new WINNER state here. That isn't too bad...

... but then, you'd have to add a new conditional in every single method to handle the WINNER state; that's a lot of code to modify.

turnCrank() will get especially messy, because you'd have to add code to check to see whether you've got a WINNER and then switch to either the WINNER state or the SOLD state.

Penerapan State Pattern

- Karena semua kode sebelumnya *belum* menerapkan **desain pattern**…
- Kita butuh **STATE PATTERN**

1. **First, we're going to define a State interface that contains a method for every action in the Gumball Machine.**
2. **Then we're going to implement a State class for every state of the machine. These classes will be responsible for the behavior of the machine when it is in the corresponding state.**
3. **Finally, we're going to get rid of all of our conditional code and instead delegate to the state class to do the work for us.**

State Pattern

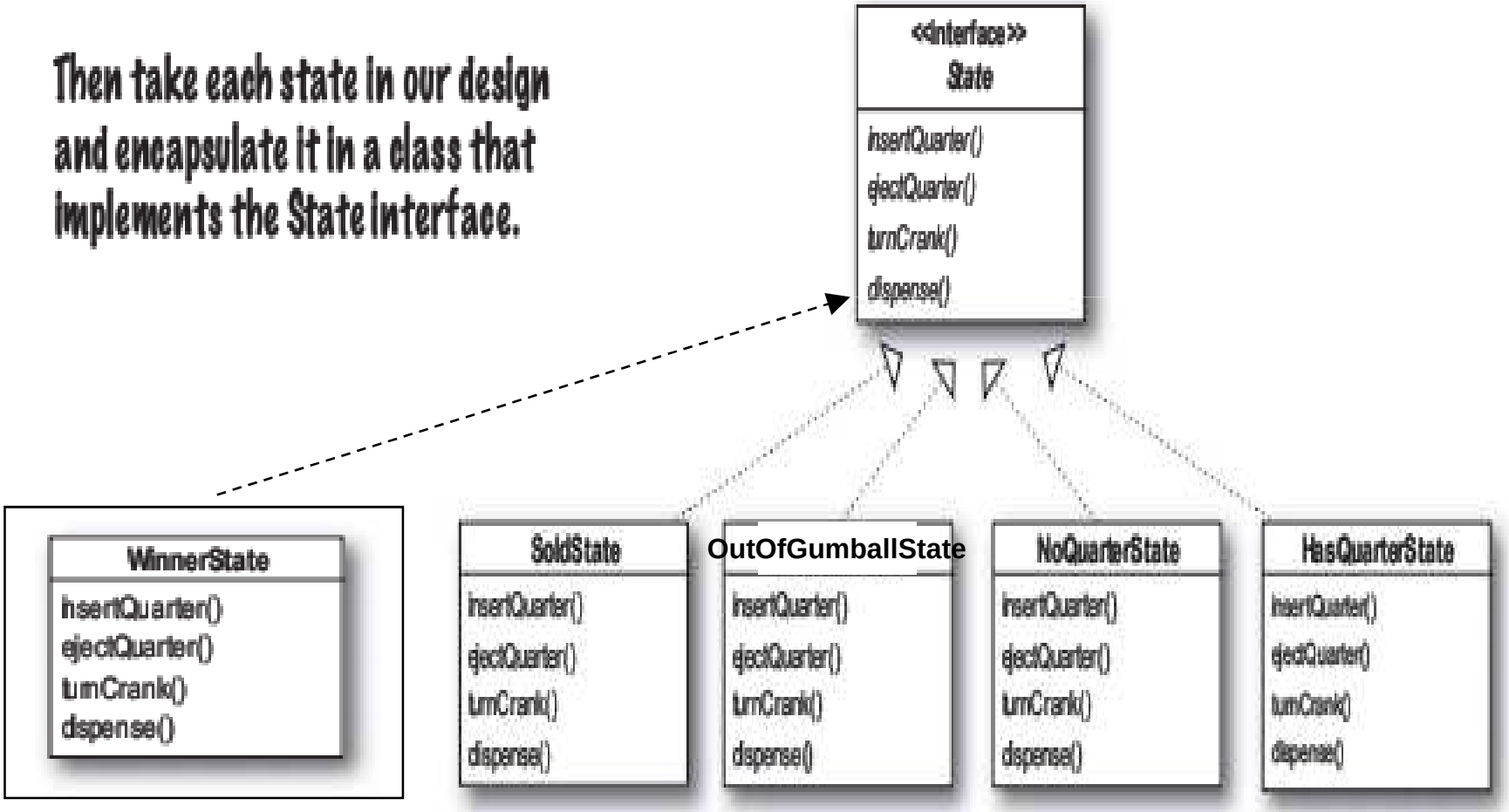
- “Allows an object to **alter** it’s behavior when its ***internal state changes***.
- The object will appear to **change** it’s class”
- Perubahan State terjadi di dalam Class State masing-masing sesuai dengan methodnya

State Pattern

- State pattern sangat berguna jika ada suatu objek yang dapat berada dalam beberapa **state**, dimana objek tersebut memiliki **behaviour** yang berbeda untuk masing-masing state
- State pattern **menyederhanakan** aksi yang memiliki statement kondisional yang besar yang tergantung pada state obyek
  - Menggantikan **switch** atau **if** biasa

Class diagram kasus

Then take each state in our design and encapsulate it in a class that implements the State interface.



Langkah-langkah

- Buat interface untuk State
- Buat class untuk masing-masing State
  - Outofgumball state
  - Soldstate
  - Noquarterstate
  - Winnerstate
  - Hasquarterstate
- Buat class GumballMachine
- Test program

Interface **State**

```
public interface State {
    //actions
    public void insertQuarter();
    public void ejectQuarter();
    public void turnCrank();
    public void dispense();
}
```

OutOfGumballState (gumball habis)

```
public class OutOfGumballState implements State {
    GumballMachine gumballMachine;

  public OutOfGumballState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
  }

  public void insertQuarter() {
    System.out.println("You can't insert a quarter, the machine is out of gumball");
  }

  public void ejectQuarter() {
    System.out.println("You can't eject, you haven't inserted a quarter yet");
  }

  public void turnCrank() {
    System.out.println("You turned, but there are no gumballs");
  }

  public void dispense() {
    System.out.println("No gumball dispensed");
  }

  public String toString() {
    return "out of gumball";
  }
}
```

SoldState (gumball keluar)

```
public class SoldState implements State {

    GumballMachine gumballMachine;

    public SoldState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

  public void insertQuarter() {
    System.out.println("Please wait, we're already giving you a gumball");
  }

  public void ejectQuarter() {
    System.out.println("Sorry, you already turned the crank");
  }

  public void turnCrank() {
    System.out.println("Turning twice doesn't get you another gumball!");
  }

  public void dispense() {
    gumballMachine.releaseBall();
    if (gumballMachine.getCount() > 0) {
      gumballMachine.setState(gumballMachine.getNoQuarterState());
    } else {
      System.out.println("Oops, out of gumballs!");
      gumballMachine.setState(gumballMachine.getOutOfGumballState());
    }
  }

  public String toString() {
    return "dispensing a gumball";
  }
}
```

Perubahan State

NoQuarterState (mesin tdk ada uang)

```
public class NoQuarterState implements State {
    GumballMachine gumballMachine;

    public NoQuarterState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

  public void insertQuarter() {
    System.out.println("You inserted a quarter");
    gumballMachine.setState(gumballMachine.getHasQuarterState());
  }

  public void ejectQuarter() {
    System.out.println("You haven't inserted a quarter");
  }

  public void turnCrank() {
    System.out.println("You turned, but there's no quarter");
   }

  public void dispense() {
    System.out.println("You need to pay first");
  }

  public String toString() {
    return "waiting for quarter";
  }
}
```

Perubahan State

HasQuarter (mesin ada uang)

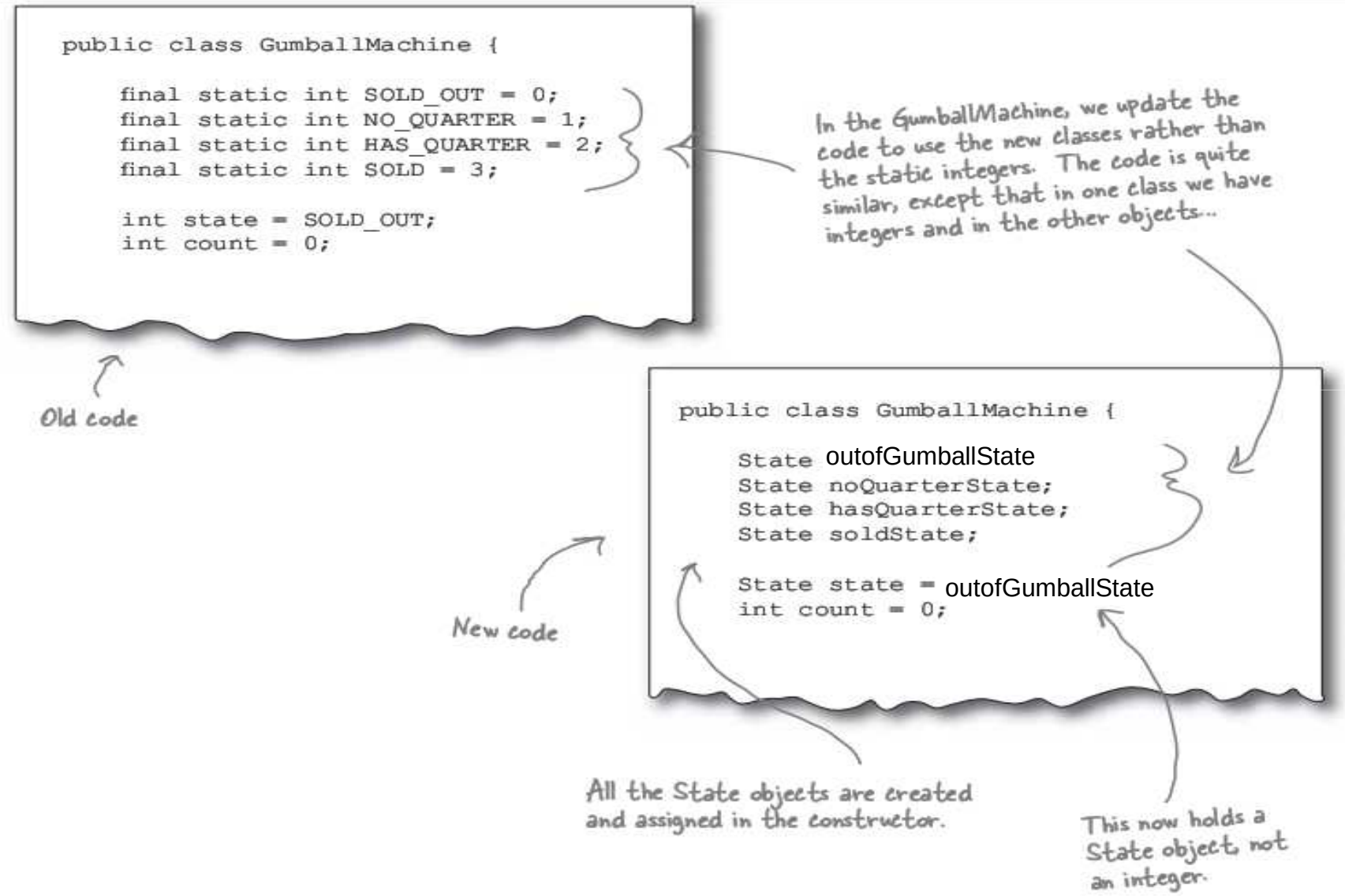
```
import java.util.Random;

public class HasQuarterState implements State {
  Random randomWinner = new Random(System.currentTimeMillis());
  GumballMachine gumballMachine;

  public HasQuarterState(GumballMachine gumballMachine) {
    this.gumballMachine = gumballMachine;
  }

  public void insertQuarter() {
    System.out.println("You can't insert another quarter");
  }

  public void ejectQuarter() {
    System.out.println("Quarter returned");
    gumballMachine.setState(gumballMachine.getNoQuarterState());
  }

  public void turnCrank() {
    //gumballMachine.setState(gumballMachine.getSoldState());
    System.out.println("You turned...");
    int winner = randomWinner.nextInt(10);
    if ((winner == 0) && (gumballMachine.getCount() > 1)) {
      gumballMachine.setState(gumballMachine.getWinnerState());
    } else {
      gumballMachine.setState(gumballMachine.getSoldState());
    }
  }

    public void dispense() {
        System.out.println("No gumball dispensed");
    }

  public String toString() {
    return "waiting for turn of crank";
  }
}
```

Perubahan State

Terakhir : WinnerState

```
public class WinnerState implements State {
    GumballMachine gumballMachine;

    public WinnerState(GumballMachine gumballMachine) {
        this.gumballMachine = gumballMachine;
    }

  public void insertQuarter() {
    System.out.println("Please wait, we're already giving you a Gumball");
  }

  public void ejectQuarter() {
    System.out.println("Please wait, we're already giving you a Gumball");
  }

  public void turnCrank() {
    System.out.println("Turning again doesn't get you another gumball!");
  }

  public void dispense() {
    System.out.println("YOU'RE A WINNER! You get two gumballs for your quarter");
    gumballMachine.releaseBall();
    if (gumballMachine.getCount() == 0) {
      gumballMachine.setState(gumballMachine.getOutOfGumballState());
    } else {
      gumballMachine.releaseBall();
      if (gumballMachine.getCount() > 0) {
        gumballMachine.setState(gumballMachine.getNoQuarterState());
      } else {
            System.out.println("Oops, out of gumballs!");
        gumballMachine.setState(gumballMachine.getOutOfGumballState());
      }
    }
  }

  public String toString() {
    return "despensing two gumballs for your quarter, because YOU'RE A WINNER!";
  }
}
```

Rewriting GumballMachine



GumballMachine

```
public class GumballMachine {

   State outofGumballState;
   State noQuarterState;
   State hasQuarterState;
   State soldState;
   State winnerState;

   State state = outofGumballState;
   int count = 0;

   public GumballMachine(int numberGumballs) {
      outofGumballState = new OutofGumballState(this);
      noQuarterState = new NoQuarterState(this);
      hasQuarterState = new HasQuarterState(this);
      soldState = new SoldState(this);
      winnerState = new WinnerState(this);

      this.count = numberGumballs;
      if (numberGumballs > 0) {
         state = noQuarterState;
      }
   }

   public void insertQuarter() {
      state.insertQuarter();
   }

   public void ejectQuarter() {
      state.ejectQuarter();
   }
```

Inisialisasi State

```
public void turnCrank() {
  state.turnCrank();
  state.dispense();
}

void setState(State state) {
  this.state = state;
}

void releaseBall() {
  System.out.println("A gumball comes rolling out the slot...");
  if (count != 0) {
    count = count - 1;
  }
}

int getCount() {
  return count;
}

void refill(int count) {
  this.count = count;
  state = noQuarterState;
}

  public State getState() {
      return state;
  }

  public State getOutOfGumballState() {
      return outOfGumballState;
  }

  public State getNoQuarterState() {
      return noQuarterState;
  }

  public State getHasQuarterState() {
      return hasQuarterState;
  }
```

lanjutan

```
    public State getSoldState() {
        return soldState;
    }

    public State getWinnerState() {
        return winnerState;
    }

  public String toString() {
    StringBuffer result = new StringBuffer();
    result.append("\nMighty Gumball, Inc.");
    result.append("\nJava-enabled Standing Gumball Model #2004");
    result.append("\nInventory: " + count + " gumball");
    if (count != 1) {
      result.append("s");
    }
    result.append("\n");
    result.append("Machine is " + state + "\n");
    return result.toString();
  }
}
```

Keuntungan

- Localizes **all behavior** associated with a particular **state** into **one object**. (= State)
  - New **state** and transitions can be **added** easily by defining **new subclasses**.
  - **Simplifies** maintenance.
- It makes **state transitions explicit**.
  - Separate objects for separate states makes transition explicit rather than using **internal data structure values** to define transitions in one combined object.
  - Using **setState**
- State objects **can be shared**.
  - Methods can share **State** objects if there are no instance variables.

GumballMachineTestDrive

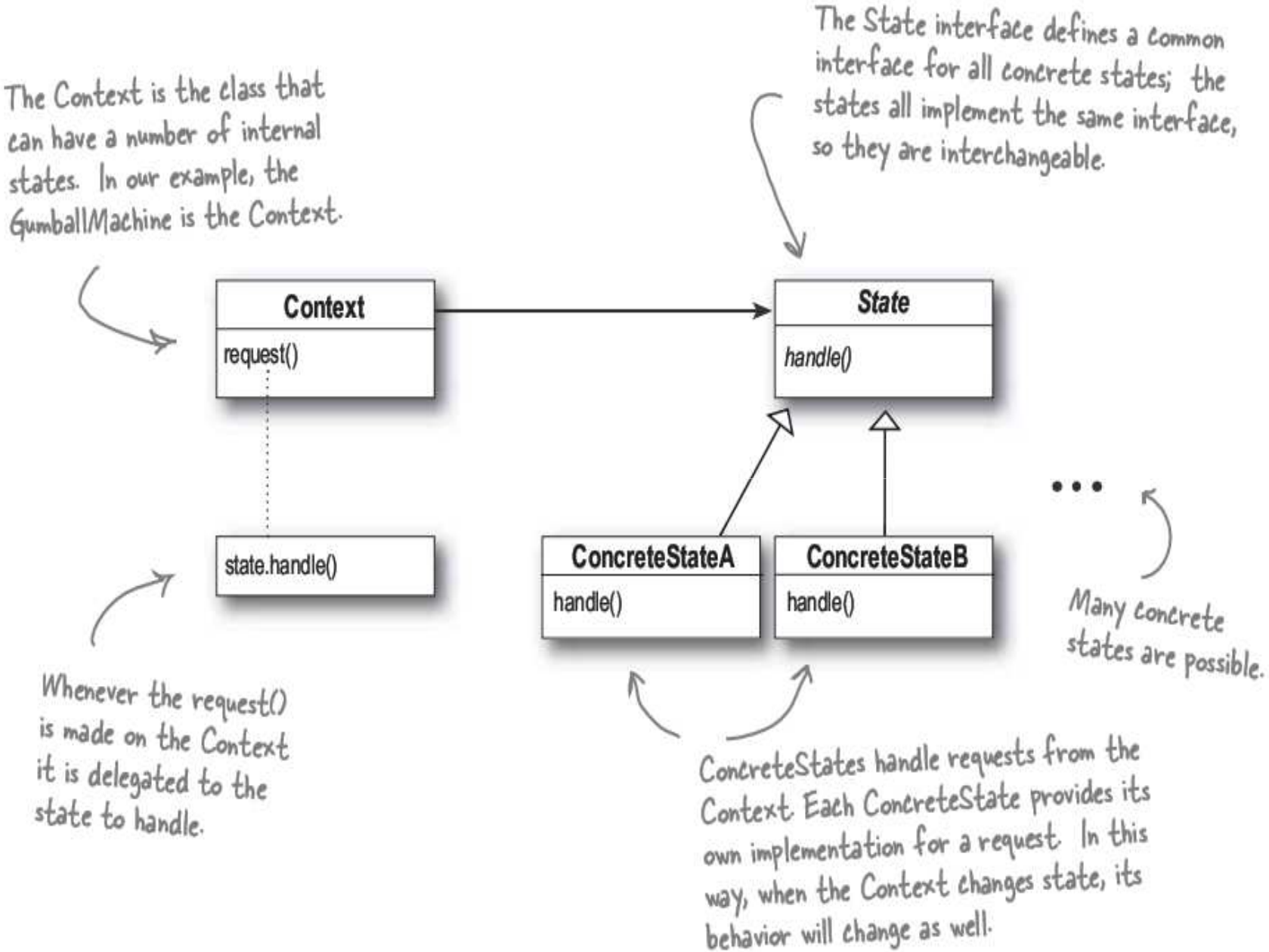
```
public class GumballMachineTestDrive2 {

    public static void main(String[] args) {
        GumballMachine gumballMachine = new GumballMachine(5);

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

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

PERCOBAAN PERTAMA!

```
D:\Dosen\PBK\program\bab6\gumballstate>java -cp . GumballMachineTestDrive2

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
You inserted a quarter
You turned...
YOU'RE A WINNER! You get two gumballs for your quarter
A gumball comes rolling out the slot...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 1 gumball
Machine is waiting for quarter
```

PERCOBAAN KEDUA!

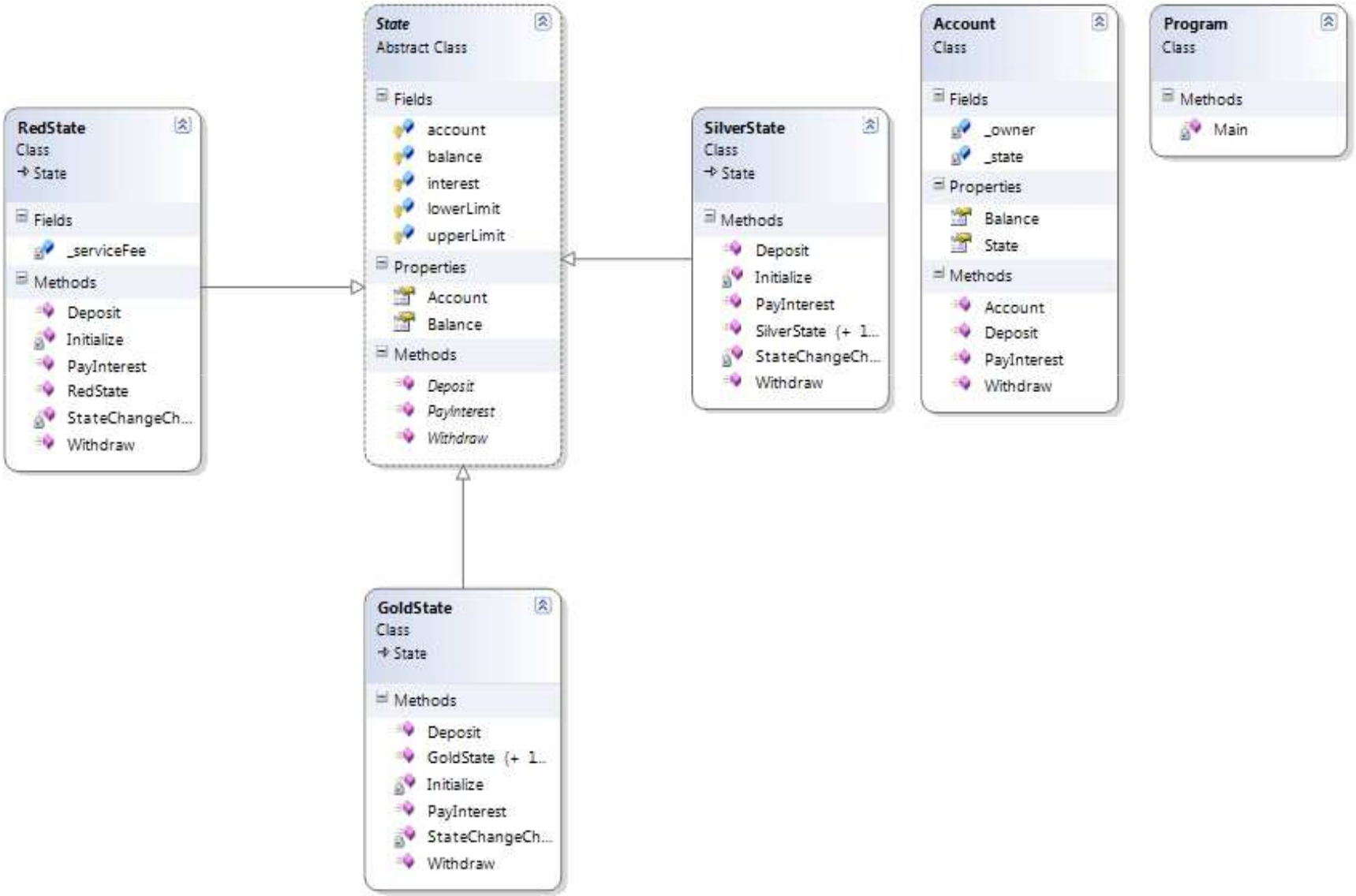
```
D:\Dosen\PBK\program\bab6\gumballstate>java -cp . GumballMachineTestDrive2

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 5 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 4 gumballs
Machine is waiting for quarter

You inserted a quarter
You turned...
A gumball comes rolling out the slot...
You inserted a quarter
You turned...
A gumball comes rolling out the slot...

Mighty Gumball, Inc.
Java-enabled Standing Gumball Model #2004
Inventory: 2 gumballs
Machine is waiting for quarter
```


The State interface defines a common interface for all concrete states; the states all implement the same interface, so they are interchangeable.
The Context is the class that can have a number of internal states. In our example, the GumballMachine is the Context.
Context
request()
State
handle()
state.handle()
ConcreteStateA
handle()
ConcreteStateB
handle()
Many concrete states are possible.
Whenever the request() is made on the Context it is delegated to the state to handle.
ConcreteStates handle requests from the Context. Each ConcreteState provides its own implementation for a request. In this way, when the Context changes state, its behavior will change as well.

Contoh lain: Account Bank

- Setiap nasabah memiliki buku tabungan
  - Bisa cek saldo, ambil, bayar pajak tabungan, dan nabung
- Pertama buku tabungan akan memiliki state “Silver”
  - Kondisi **balance** = 0
- State:
  - **Silver**: balance = 0, lower = 0, upper = 1000
  - **Red**: pajak = 0, lower = -100, biaya = 15, upper = 0
  - **Gold**: pajak = 0.05, lower = 1000, upper = 10000000

Contoh lain: Account Bank



Demo

- AccountBank

NEXT

- TAS
- Essay + Pil Ganda
- Open 1 lembar kertas bolak balik boleh diketik
- Materi dari setelah TTS