State Pattern

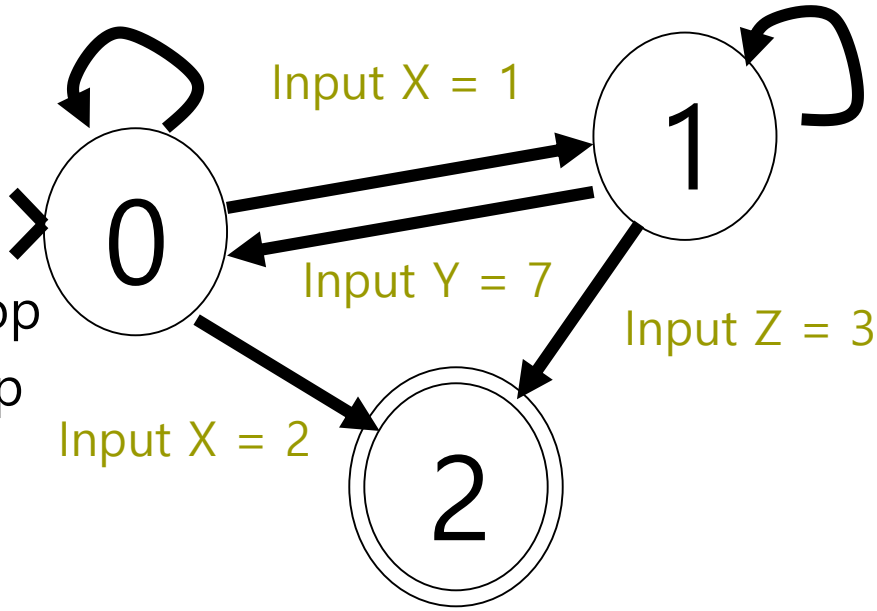
514770-1
Fall 2023
11/21/2023
Kyoung Shin Park
Computer Engineering
Dankook University

State Pattern

- "Allow an object to alter its behavior when its internal state changes. The object will appear to change its class."
- Also known as "**Objects for States**"
- An **object-oriented state machine**
- The State pattern is used when an object changes its behavior based on its internal state.
- In State pattern we create objects which represent various **states** and a **context** object whose behavior varies as its state object changes.
- The State pattern is closely related to the concept of a **Finite State Machine**.

Finite State Machine

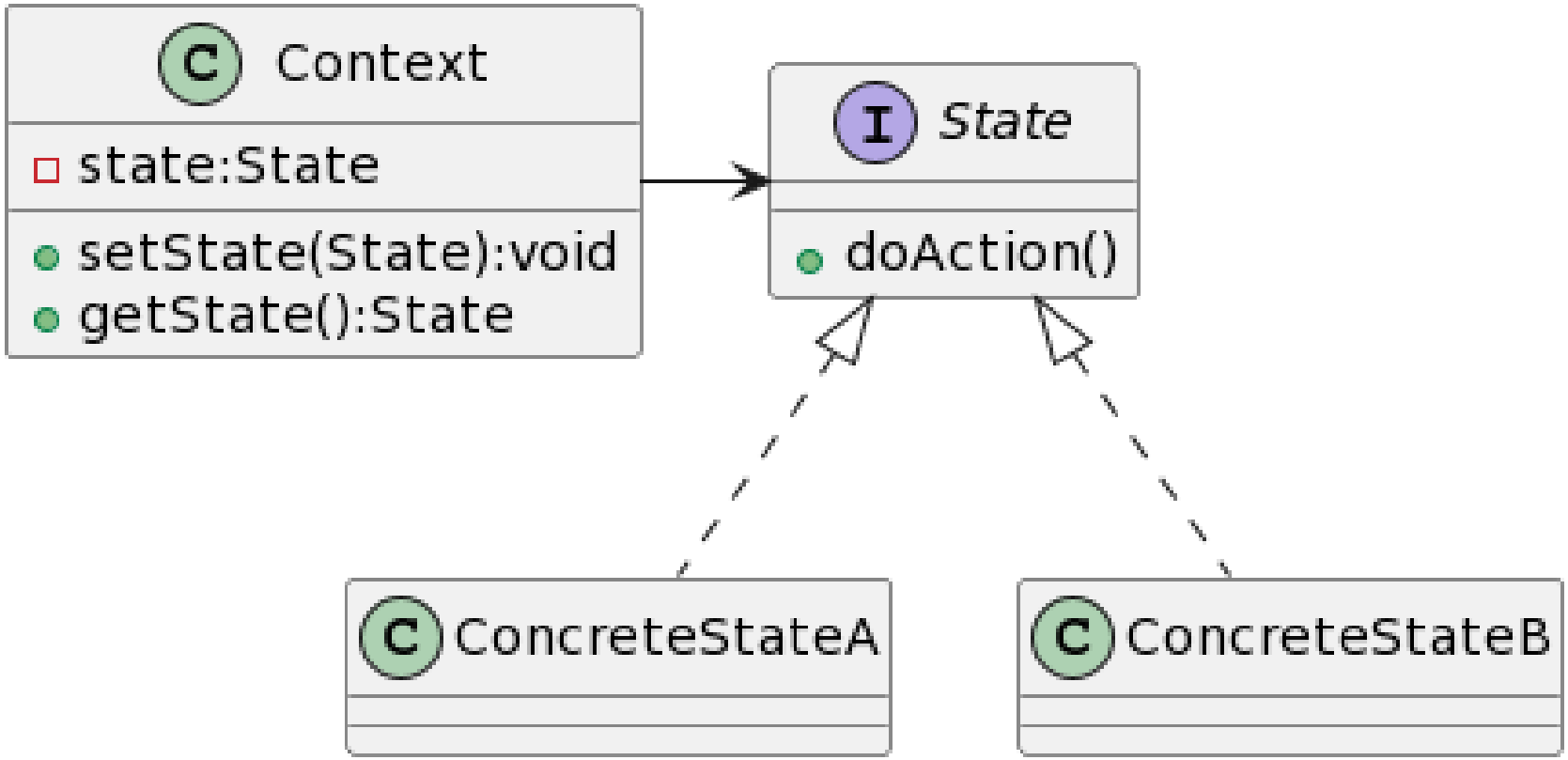
- **Finite State Machine (FSM)** or **Finite Automata**, or simply a **state machine**.
- An FSM is defined by a list of its states, its initial state, and the inputs that trigger each transition.
  - States
  - Inputs
  - Transitions
- For example,
  - Game character: walk, run, stop
  - Electronic goods: on, off, sleep
  - Turnstile: locked, unlocked



State Pattern

	Description
Pattern	State
Problem	State machines are usually implemented with lots of conditional operators (**if** or **switch**) that select the appropriate behavior depending on the current state of the object.
Solution	The State pattern allows the object for changing its behavior without changing its class.
Result	Single Responsibility Principle, Open/Closed Principle, Cleaner and more maintainable code

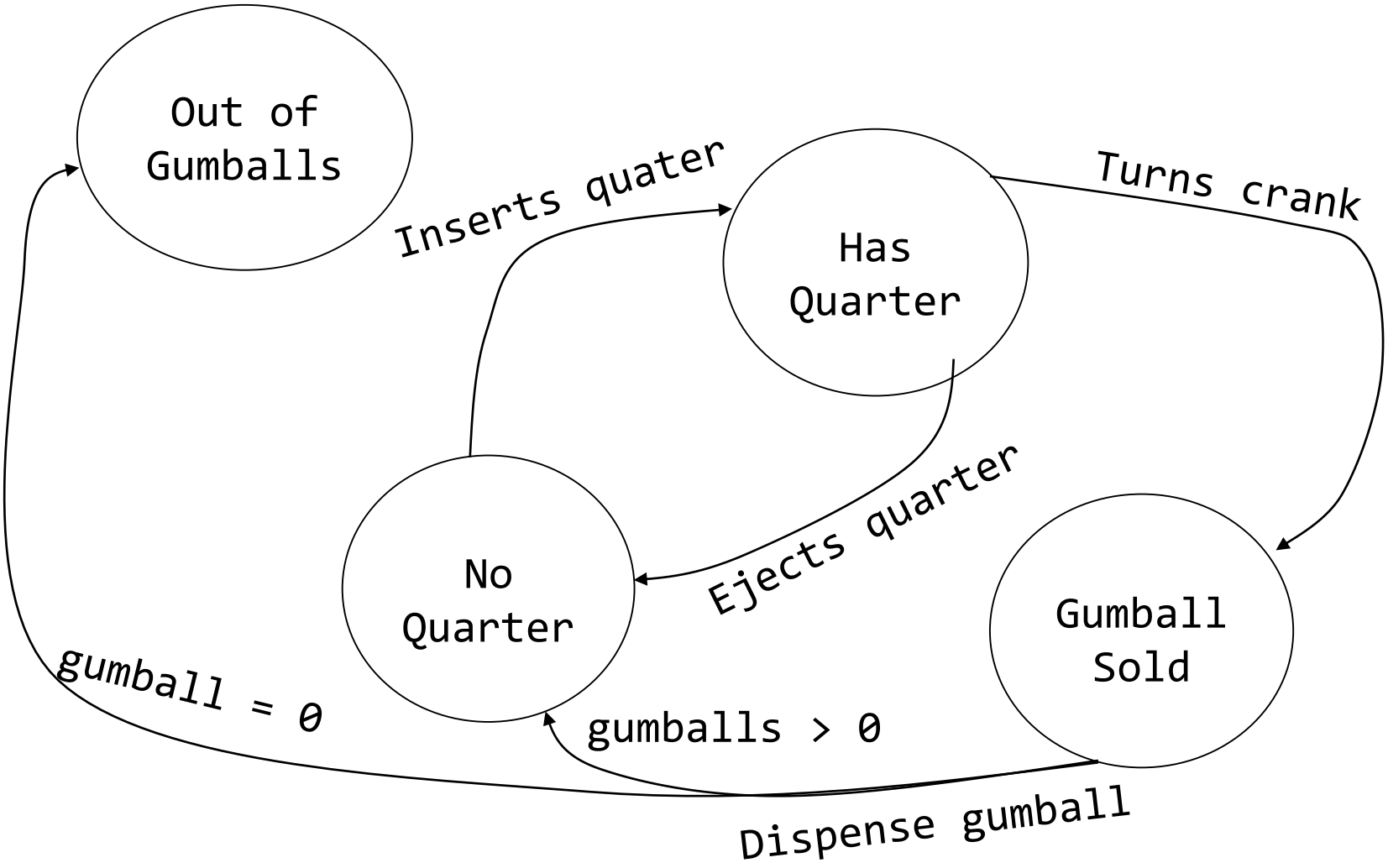
State Pattern



Define State Pattern

- Context
  - Context stores **a reference to one of the concrete state objects** and delegates to it all state-specific work. Context communicates with the state object via the state interface. Context exposes a **setter for passing it a new state object**.
- State
  - The State interface declares the **state-specific methods** (**what each concrete state should do**).
- ConcreteStateA, ConcreteStateB
  - They provide their own **implementations for state-specific methods**. To avoid duplication of similar code across multiple states, you may provide intermediate abstract classes that encapsulate some common behavior.

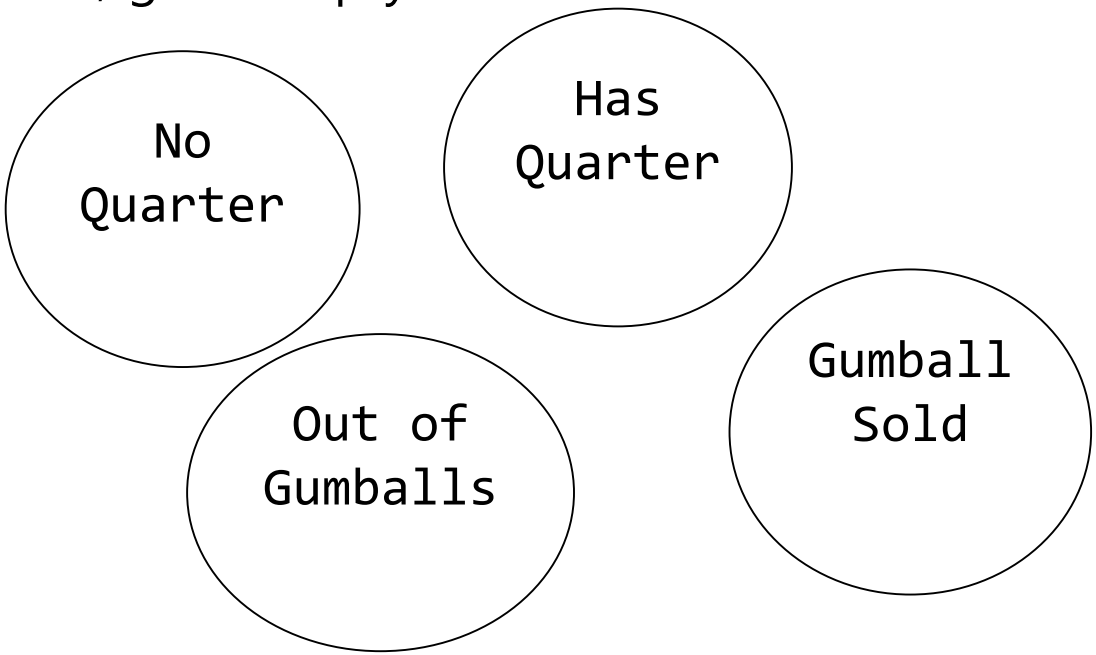
Gumball Machine (HFDP Ch. 10)



Finite State Machine

Gumball Machine (HFDP Ch. 10)

- Implementing **state machines**
  - First, gather up your states:



Gumball Machine (HFDP Ch. 10)

- Next, create an instance variable to hold the current state, and define values for each of the states:

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

int state = SOLD_OUT;
```

- Now, gather up all the actions that can happen in the system.

Ejects quarter

Inserts quarter

Turns crank

dispense

Gumball Machine (HFDP Ch. 10)

- Now, create a class that acts as the state machine.

```
public class GumballMachine {
  final static int SOLD_OUT = 0;
  final static int NO_QUARTER = 1;
  final static int HAS_QUARTER = 2;
  final static int SOLD = 3;

  int state = SOLD_OUT;
  int count = 0;

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

Gumball Machine (HFDP Ch. 10)

- Implement the actions as methods.

```
public void insertQuarter() {
  if (state == HAS_QUARTER) {
    System.out.println("You can’t insert another
quarter.");
  } else if (state == SOLD_OUT) {
    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.");
  } else if (state = NO_QUARTER) {
    state = HAS_QUARTER;
    System.out.println("You inserted a quarter.");
  }
}
```

Gumball Machine (HFDP Ch. 10)

```
  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 = SOLD_OUT) {
      System.out.println("You can't eject, you
haven't inserted a quarter yet. ");
    }
  }
```

Gumball Machine (HFDP Ch. 10)

```
  public void turnCrank() {
    if (state == SOLD) {
      System.out.println("Turing 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 == SOLD_OUT) {
      System.out.println("You turned, but there are
no gumballs.");
    } else if (state = HAS_QUARTER) {
      System.out.println("You turned..");
      state = SOLD;
      dispense();
    }
  }
```

```
  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 = SOLD_OUT;
      } else {
        state = NO_QUARTER;
      }
    } else if (state == NO_QUARTER) {
      System.out.println("You need to pay first.");
    } else if (state == SOLD_OUT) {
      System.out.println("No gumball dispensed.");
    } else if (state = HAS_QUARTER) {
      System.out.println("No gumball dispensed.");
    }
  }
  // other methods..
}
```

Gumball Machine (HFDP Ch. 10)

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

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

Gumball Machine (HFDP Ch. 10)

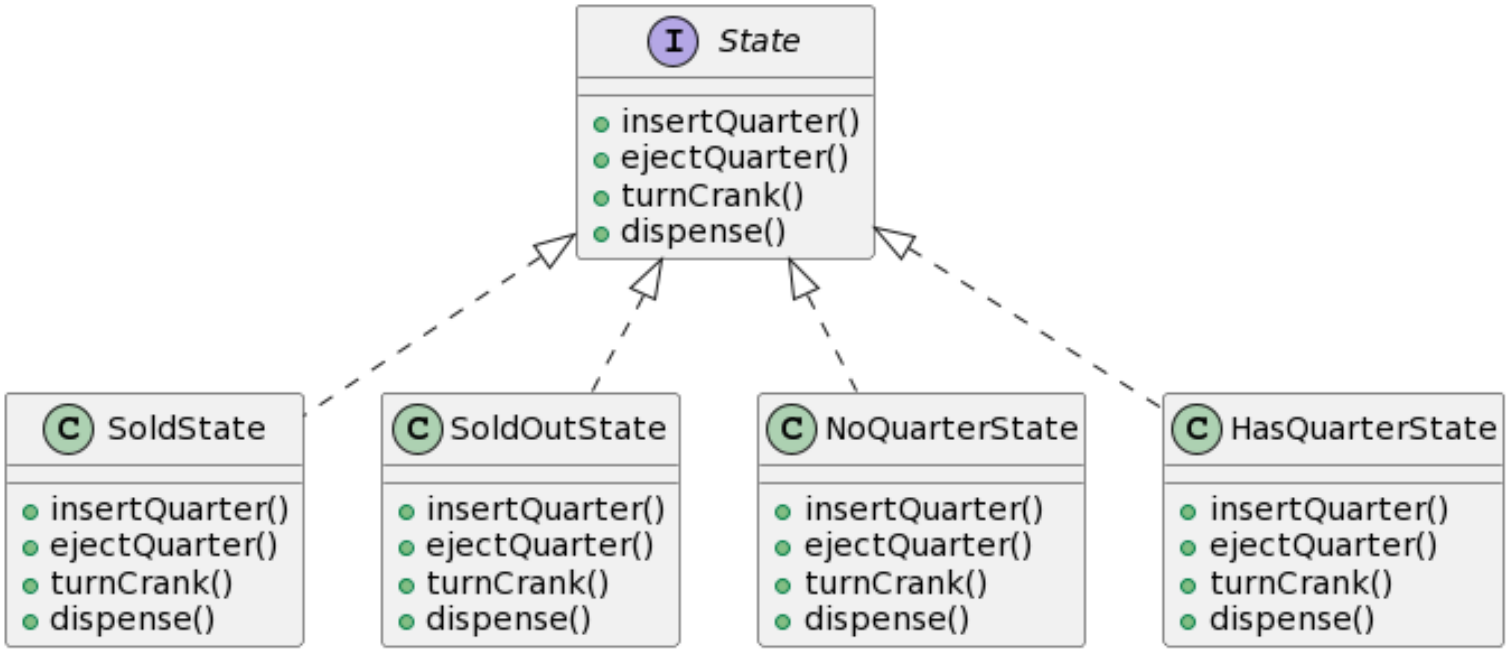
```
        gumballMachine.insertQuarter();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();
        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

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

Gumball Machine (HFDP Ch. 10)

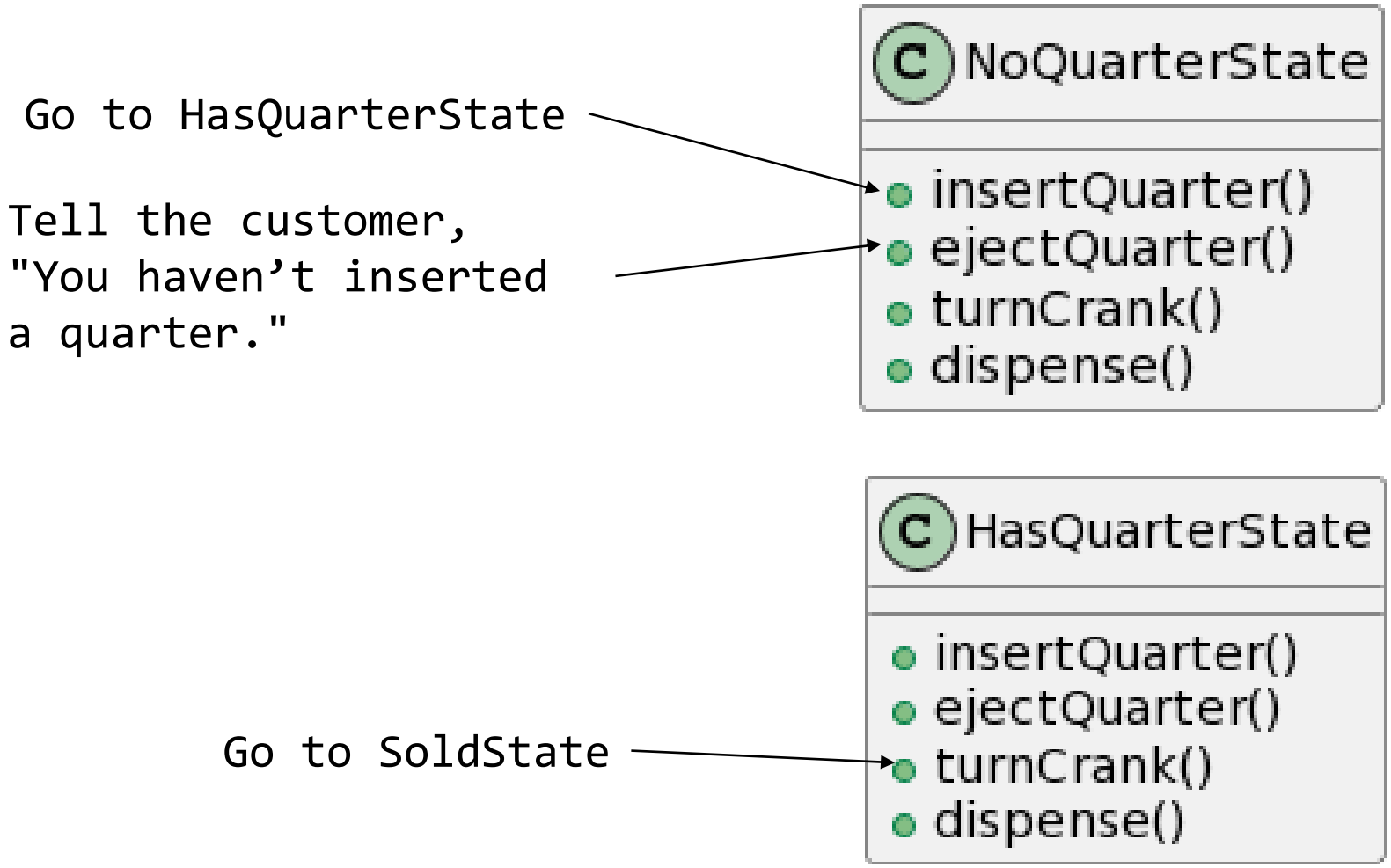
- A change request
  - 10% of the time, when the crank is turned, the customer gets two gumballs instead of one.
    - **Be a WINNER!** One in ten get a free gumball.
    - First, you'd have to add a new **WINNER state**.
    - .. But then, you'd have to add a new **conditional** in **every single method** (insertQuater, ejectQuarter, dispense) 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.

Gumball Machine (HFDP Ch. 10)

- The new design
  - First, define a **State interface** that contains a method for every action in the Gumball Machine.
  - Then, **implement a State class** for every state of the machine.
  - Finally, get rid of all of our conditional code and instead delegate to the state class to do the work for us.

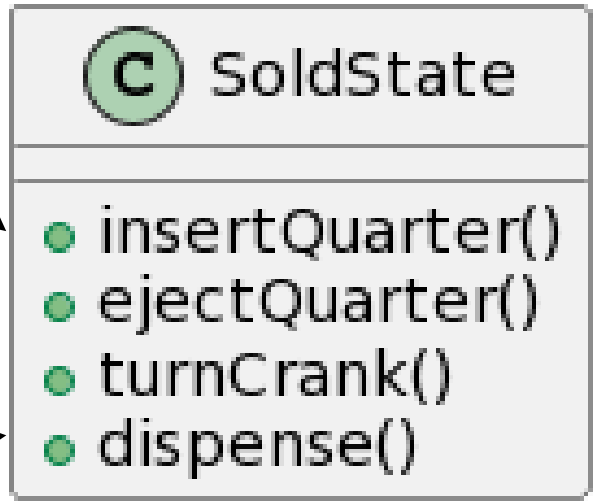


Gumball Machine (HFDP Ch. 10)

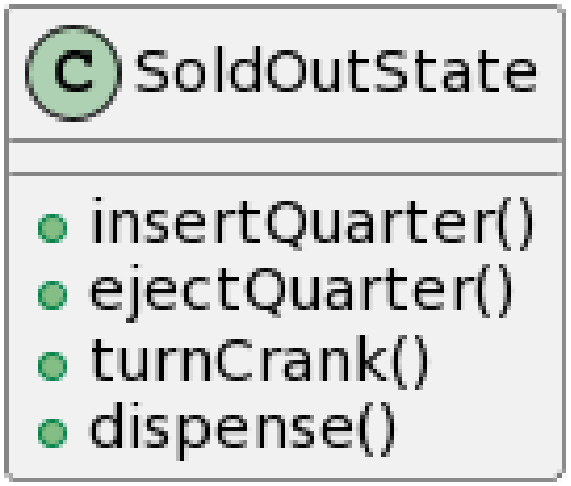


Gumball Machine (HFDP Ch. 10)

Tell the customer,
"Please wait, we're already
giving you a gumball."



Dispense one gumball. Check
number of gumballs; if > 0, go
to NoQuarterState, otherwise,
go to SoldOutState.



Tell the customer,
"There are no gumballs."

Gumball Machine (HFDP Ch. 10)

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

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

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

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

Gumball Machine (HFDP Ch. 10)

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

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

Gumball Machine (HFDP Ch. 10)

- Reworking the Gumball Machine
  - Switch the code from the state related instance variables using integers to **using state objects**.

```
public class GumballMachine {
  State soldOutState;
  State noQuarterState;
  State hasQuarterState;
  State soldState;

  State state = soldOutState;
  int count = 0;

  public GumballMachine(int numberGumballs) {
    soldOutState = new SoldOutState(this);
    noQuarterState = new NoQuarterState(this);
    hasQuarterState = new HasQuarterState(this);
    soldState = new SoldState(this);
```

Gumball Machine (HFDP Ch. 10)

```
    this.count = numberGumballs;
    if (numberGumballs > 0 ) {
      state = noQuarterState;
    }
  }
  public void insertQuarter() {
    state.insertQuarter();
  }
  public void ejectQuarter() {
    state.ejectQuarter();
  }
  public void turnCrank() {
    state.turnCrank();
    state.dispense();
  }
  void setState(State state) {
    this.state = state;
  }
```

Gumball Machine (HFDP Ch. 10)

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

  // more methods including getters for each State
}
```

Gumball Machine (HFDP Ch. 10)

- Implementing HasQuarterState

```
public class HasQuarterState implements State {
  GumballMachine gm;

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

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

  public void ejectQuarter() {
    System.out.println("Quarter returned.");
    gm.setState(gm.getNoQuaterState());
  }
```

Gumball Machine (HFDP Ch. 10)

```
    public void turnCrank() {
        System.out.println("You turned..");
        gm.setState(gm.getSoldState());
    }

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

Gumball Machine (HFDP Ch. 10)

- Implementing SoldState

```
public class SoldState implements State {
  GumballMachine gm;

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

  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.");
  }
```

Gumball Machine (HFDP Ch. 10)

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

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

Gumball Machine (HFDP Ch. 10)

- Implementing SoldOutState

```
public class SoldOutState implements State {
  GumballMachine gm;

  public SoldOutState(GumballMachine gm) {
    gm = gm;
  }

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

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

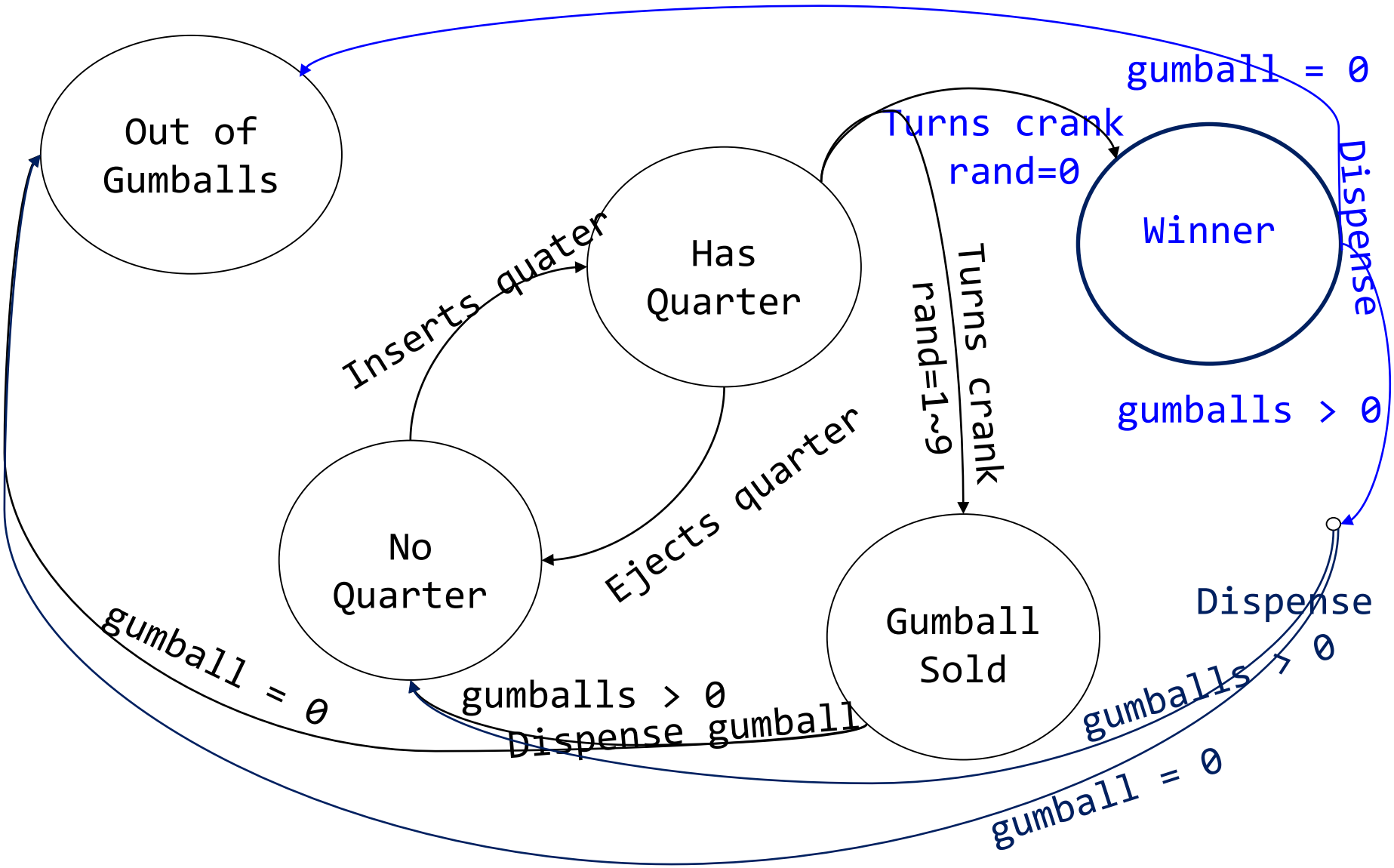
Gumball Machine (HFDP Ch. 10)

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

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

Gumball Machine (HFDP Ch. 10)

- In State pattern, states are class.
- It gets rid of if-statements.
- State machine is open to extensions that add new state classes, such as Winner State.

Gumball Machine (HFDP Ch. 10)



Gumball Machine (HFDP Ch. 10)

- To make a gumball machine that gives you an extra gumball every ten times

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

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

  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() {
    gm.releaseBall();
    if (gm.getCount() == 0) {
      gm.setState(gm.getSoldOutState());
    } else {
      gm.releaseBall();
      System.out.println("YOU'RE A WINNER! You got
two gumballs for your quarter.");
      if (gm.getCount() > 0) {
        gm.setState(gm.getNoQuarterState());
      } else {
        System.out.println("Oops, out of gumballs!");
        gm.setState(gm.getSoldOutState());
      }
    }
  }
}
```

Gumball Machine (HFDP Ch. 10)

- Reworking HasQuarterState

```
public class HasQuarterState implements State {
   Random random = new Random(
                              System.currentImeMillis());
   public void turnCrank() {
      System.out.println("You turned...");
      int winner = random.nextInt(10);
      if ((winner == 0)
                  && (gumballMachine.getCount() > 1)) {
         gumballMachine.setState(
            gumballMachine.getWinnerState());
      } else {
         gumballMachine.setState(
            gumballMachine.getSoldState());
      }
   }
}
```