Monday April 1

Lecture 22

Wedn. Exam .

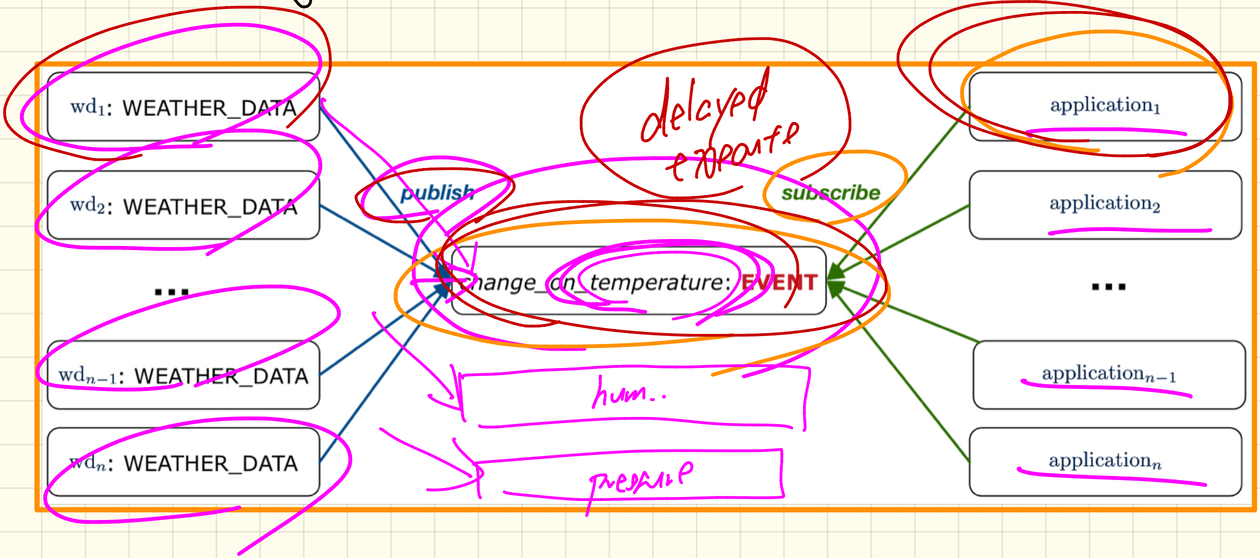
2pm

Fri. April 5 makeup class .

review

Fri April 12 .

Event-Driven Design: Multiple Subjects and Observers



wd_1: WEATHER_DATA

wd_2: WEATHER_DATA

...

wd_{n-1}: WEATHER_DATA

wd_n: WEATHER_DATA

publish

change_on_temperature: EVENT

subscribe

$\mathrm{application}_1$

$\mathrm{application}_2$

...

$\mathrm{application}_{n-1}$

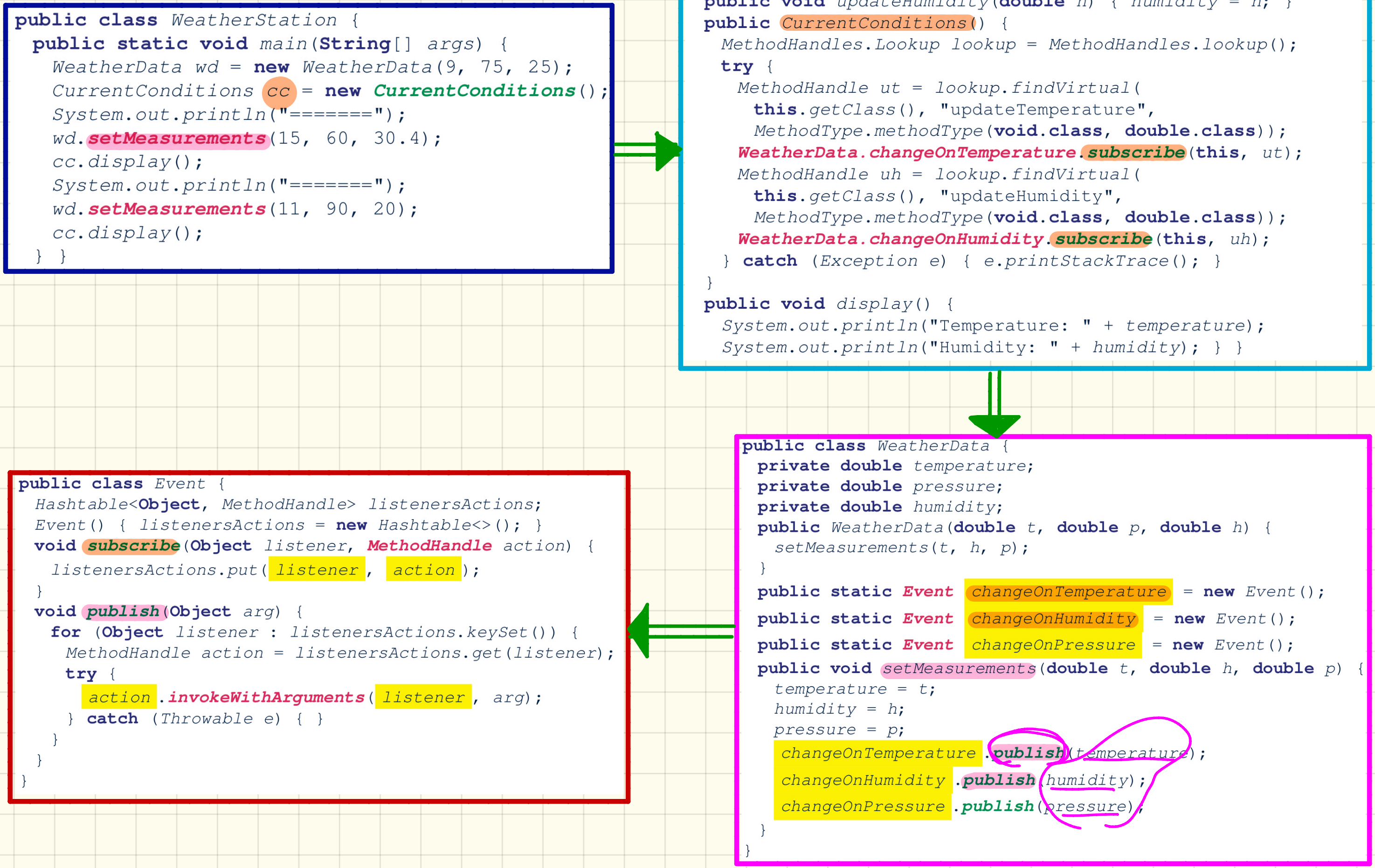
$\mathrm{application}_n$

Complexity ?

Adding a new subject? Adding a new observer?

Adding a new event type ?

Event-Driven Design in Java



```java
public class WeatherStation {
  public static void main(String[] args) {
    WeatherData wd = new WeatherData(9, 75, 25);
    CurrentConditions cc = new CurrentConditions();
    System.out.println("=======");
    wd.setMeasurements(15, 60, 30.4);
    cc.display();
    System.out.println("=======");
    wd.setMeasurements(11, 90, 20);
    cc.display();
} }
```

```java
public class CurrentConditions {
  private double temperature; private double humidity;
  public void updateTemperature(double t) { temperature = t; }
  public void updateHumidity(double h) { humidity = h; }
  public CurrentConditions() {
    MethodHandles.Lookup lookup = MethodHandles.lookup();
    try {
      MethodHandle ut = lookup.findVirtual(
        this.getClass(), "updateTemperature",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnTemperature.subscribe(this, ut);
      MethodHandle uh = lookup.findVirtual(
        this.getClass(), "updateHumidity",
        MethodType.methodType(void.class, double.class));
      WeatherData.changeOnHumidity.subscribe(this, uh);
    } catch (Exception e) { e.printStackTrace(); }
  }
  public void display() {
    System.out.println("Temperature: " + temperature);
    System.out.println("Humidity: " + humidity); } }
```

```java
public class WeatherData {
  private double temperature;
  private double pressure;
  private double humidity;
  public WeatherData(double t, double p, double h) {
    setMeasurements(t, h, p);
  }
  public static Event changeOnTemperature = new Event();
  public static Event changeOnHumidity = new Event();
  public static Event changeOnPressure = new Event();
  public void setMeasurements(double t, double h, double p) {
    temperature = t;
    humidity = h;
    pressure = p;
    changeOnTemperature.publish(temperature);
    changeOnHumidity.publish(humidity);
    changeOnPressure.publish(pressure);
  }
}
```

```java
public class Event {
  Hashtable<Object, MethodHandle> listenersActions;
  Event() { listenersActions = new Hashtable<>(); }
  void subscribe(Object listener, MethodHandle action) {
    listenersActions.put( listener , action );
  }
  void publish(Object arg) {
    for (Object listener : listenersActions.keySet()) {
      MethodHandle action = listenersActions.get(listener);
      try {
        action .invokeWithArguments( listener , arg);
      } catch (Throwable e) { }
    }
  }
}
```

Event-Driven Design in Eiffel

```
class WEATHER_STATION create make
feature
  cc: CURRENT_CONDITIONS
  make
    do create wd.make (9, 75, 25)
       create cc.make (wd)
       wd.set_measurements (15, 60, 30.4)
       cc.display
       wd.set_measurements (11, 90, 20)
       cc.display
    end
end
```

NB pub → E ← sub obs

```
class CURRENT_CONDITIONS
create make
feature -- Initialization
  make(wd: WEATHER_DATA)
    do
      wd.change_on_temperature.subscribe (agent update_temperature)
      wd.change_on_temperature.subscribe (agent update_humidity)
    end
feature
  temperature: REAL
  humidity: REAL
  update_temperature (t: REAL) do temperature := t end
  update_humidity (h: REAL) do humidity := h end
  display do ... end
end
```

humidity

*

τ: INT.

```
class WEATHER_DATA
create make
feature -- Measurements
  temperature: REAL ; humidity: REAL ; pressure: REAL
  correct_limits(t,p,h: REAL): BOOLEAN do ... end
  make (t, p, h: REAL) do ... end
feature -- Event for data changes
  change_on_temperature : EVENT[TUPLE[REAL]]once create Result end
  change_on_humidity : EVENT[TUPLE[REAL]]once create Result end
  change_on_pressure : EVENT[TUPLE[REAL]]once create Result end
feature -- Command
  set_measurements(t, p, h: REAL)
    require correct_limits(t,p,h)
    do temperature := t ; pressure := p ; humidity := h
      change_on_temperature.publish ([t])
      change_on_humidity.publish ([p])
      change_on_pressure.publish ([h])
    end
invariant correct_limits(temperature, pressure, humidity) end
```

arg.

TUPLE[REAL, INT]

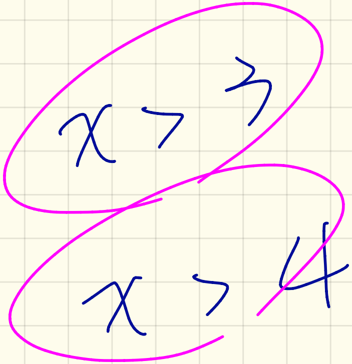
[t, τ]

Inputs for executing the stored update.

exp'd-up the stored update fn.

```
class EVENT [ARGUMENTS -> TUPLE ]
create make
feature -- Initialization
  actions: LINKED_LIST[PROCEDURE[ARGUMENTS]]
  make do create actions.make end
feature
  subscribe (an_action: PROCEDURE[ARGUMENTS])
    require action_not_already_subscribed: not actions.h
    do actions.extend (an_action)
    ensure action_subscribed: action.has(an_action) end
  publish (args: G)
    do from actions.start until actions.after
      loop actions.item.call (args) ; actions.forth end
    end
end
```

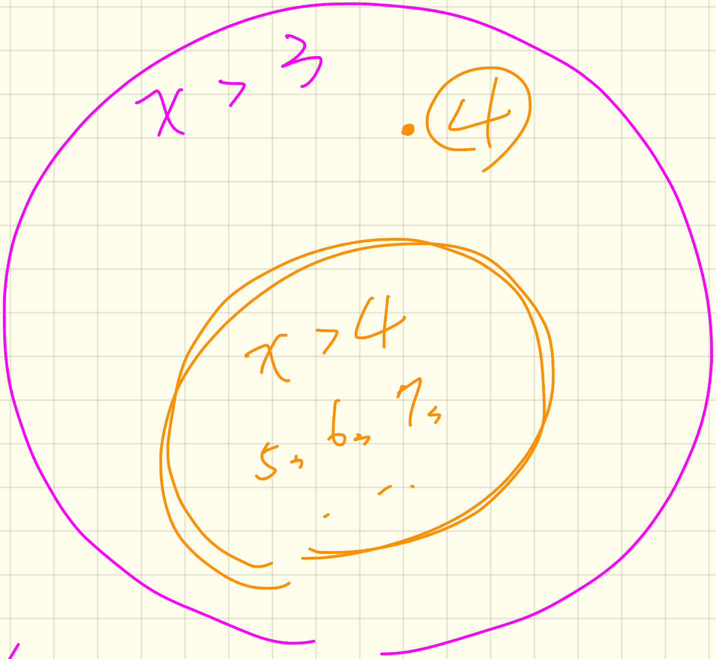
update feature input



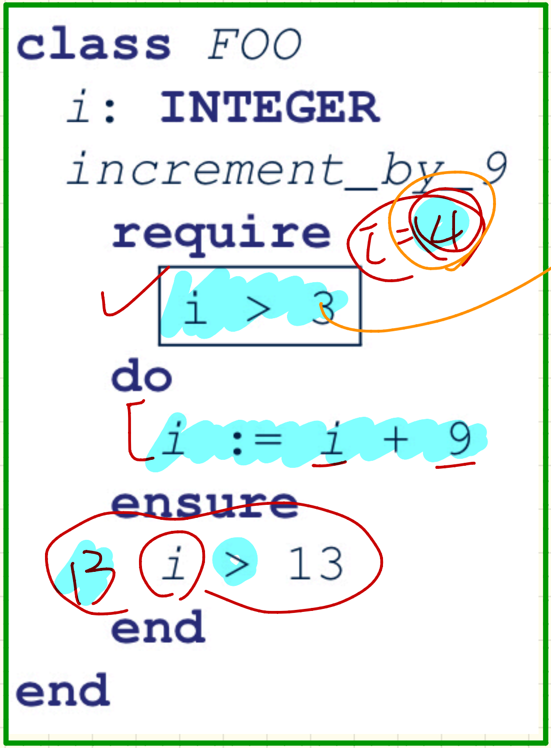
stronger

$x > 4 \Rightarrow x > 3$

weaker



Program Correctness : Example (1)



```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 3
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

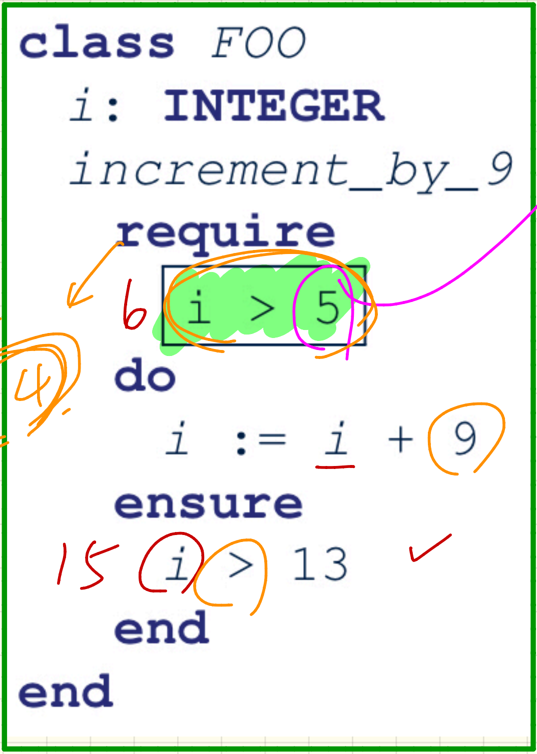
i = 14

13

too weak

```
f
  require
    ??
  do
    imp
  ensure
    Q
  end
```

Program Correctness : Example (2)



```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```

6

wp: i > 4

15 ✓

stronger: i > 6 ↓ stronger ✓ bounding

i > 7

weaker: i > 4 ✓

i > 3 ✗

appropriate: satisfying precond and executing Imp guarantees postcond.

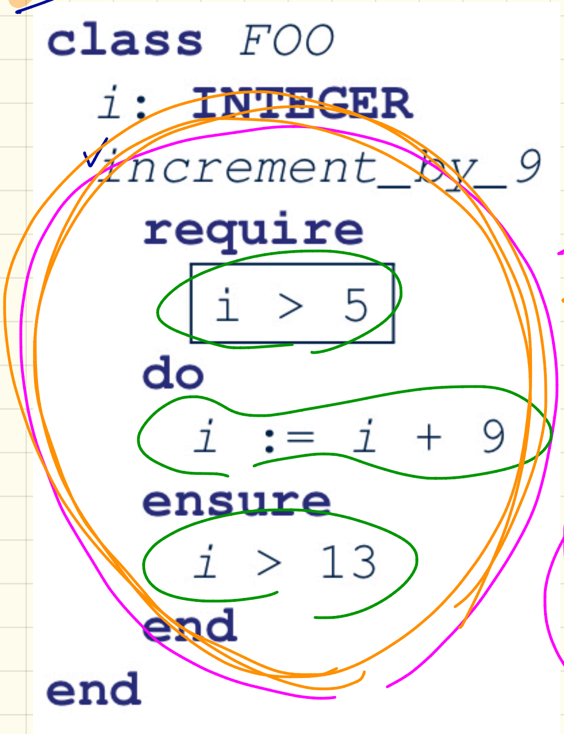
weakest precondition (wp) to establish the postcond:

1. a precondition stronger than wp ✓

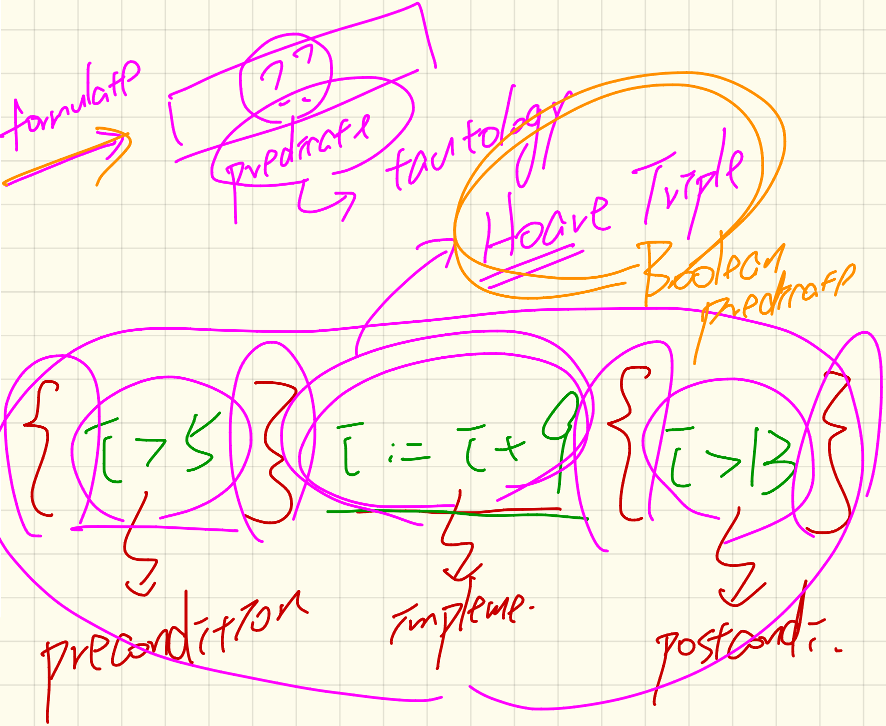
2. a precondition weaker than wp problematic -

Given.

Task: Prove that inc_by_9 is correct.



```
class FOO
  i: INTEGER
  increment_by_9
    require
      i > 5
    do
      i := i + 9
    ensure
      i > 13
    end
end
```



$$\{\underline{Q}\}\ \underline{S}\ \{\underline{R}\}$$

tautologies

$$\{i > 4\}\ i := i + 9\ \{i > 13\}$$

$$\{i > 5\}\ i := i + 9\ \{i > 13\}$$

$$\{i > 3\}\ i := i + 9\ \{i > 13\}$$

disprove:

counterexample:

$i = 4$

$$\{Q\}\ S\ \{R\}$$

(a) starting with Q and executing S will terminate.

total correctness

(b) Assume (a), does the resulting state satisfy R.

partial correctness

actual precondition

$$\{Q\}\quad S \quad \{R\}$$

$$\equiv \quad Q \Rightarrow \underline{wp}(S, R)$$

weakest precondition

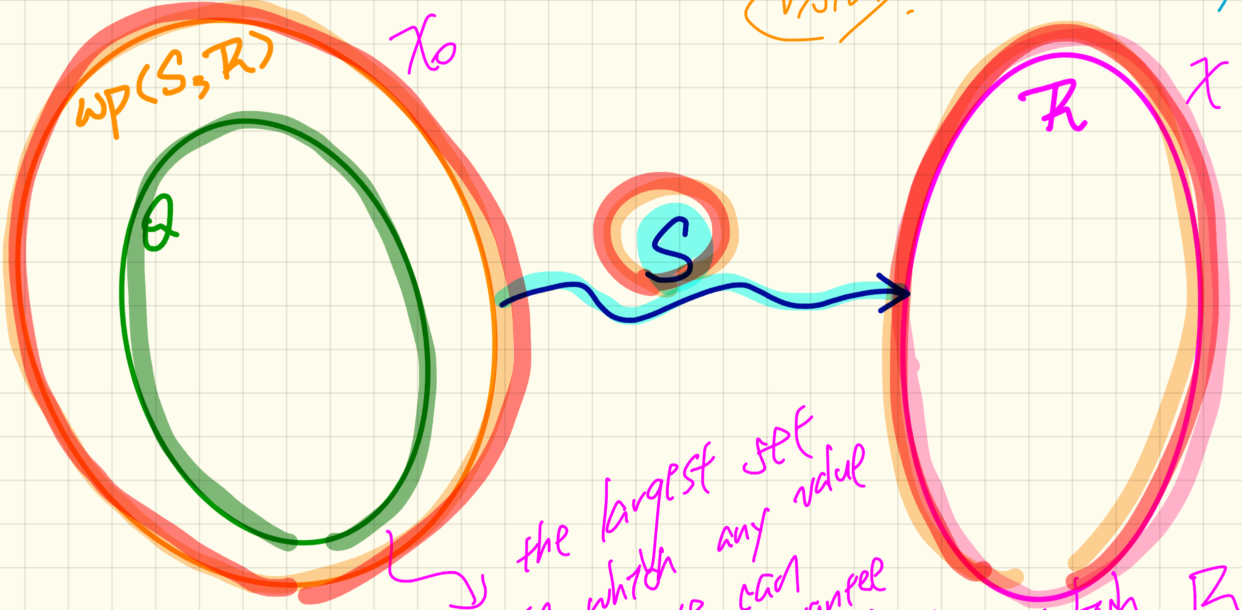
Hoare Triple as a Predicate

$$\{Q\}\ S\ \{R\} \equiv Q \Rightarrow wp(S, R)$$

$wp(S, R)$

wp → visitor.

S → program structure (composite)



the largest set in which any value there can guarantee S to establish R.

$$wp(\underbrace{x}_{x} := \underbrace{x+9}_{e}, \underbrace{x > 13}_{R})$$

$$= \underbrace{x}_{x+9} > 13 \, [\, x := \boxed{x+9} \,]$$

$$= x+9 > 13 \qquad x > 4 .$$