Subcontracting

Readings: OOSCS2 Chapters 14 – 16

EECS3311 M: Software Design
Winter 2019

CHEN-WEI WANG

Aspects of Inheritance

- ***Code Reuse***
- Substitutability
  - ○ ***Polymorphism*** and ***Dynamic Binding***

    [compile-time type checks]

  - ○ *Sub-contracting*

    [runtime behaviour checks]

Background of Logic (1)

Given **preconditions** P_1 and P_2, we say that

P_2 **requires less** than P_1 if

P_2 is **less strict** on (thus **allowing more**) inputs than P_1 does.

$$\{\, x \mid P_1(x) \,\} \subseteq \{\, x \mid P_2(x) \,\}$$

More concisely:

$$P_1 \Rightarrow P_2$$

e.g., For command `withdraw(amount: amount)`,

$P_2 : amount \geq 0$ **requires less** than $P_1 : amount > 0$

What is the **precondition** that **requires the least**? [**true**]

Background of Logic (2)

Given <mark>*postconditions*</mark> or <mark>*invariants*</mark> Q_1 and Q_2, we say that

$\boxed{Q_2 \textbf{\textit{ ensures more}} \text{ than } Q_1}$ if

Q_2 is ***stricter*** on (thus ***allowing less***) outputs than Q_1 does.

$$\{ \, x \mid Q_2(x) \, \} \subseteq \{ \, x \mid Q_1(x) \, \}$$

More concisely:

$$Q_2 \Rightarrow Q_1$$

e.g., For query `q(i: INTEGER): BOOLEAN`,

$\boxed{Q_2 : \textbf{Result} = (i > 0) \wedge (i \textbf{ mod } 2 = 0)}$ ***ensures more*** than

$\boxed{Q_1 : \textbf{Result} = (i > 0) \vee (i \textbf{ mod } 2 = 0)}$

What is the <mark>*postcondition*</mark> that ***ensures the most***? [***false***]
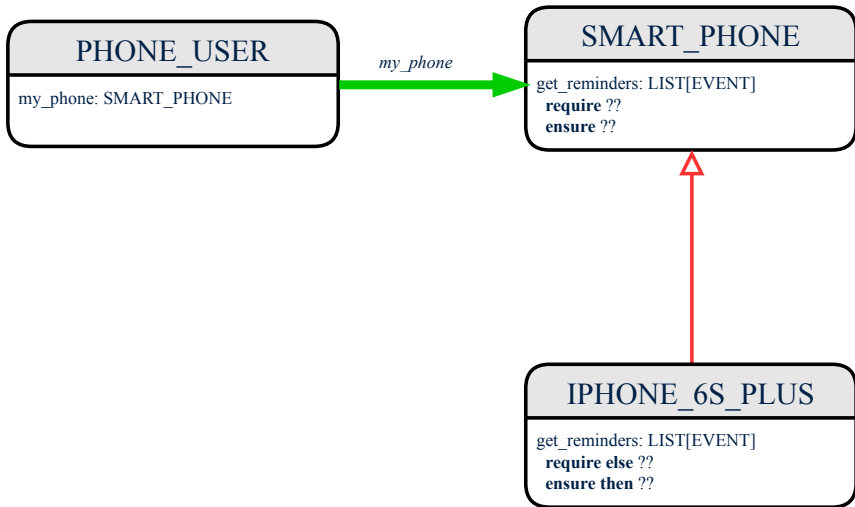
Inheritance and Contracts (1)

- The fact that we allow <mark>*polymorphism*</mark> :

```
local my_phone: SMART_PHONE
      i_phone: IPHONE_6S_PLUS
      samsung_phone: GALAXY_S6_EDGE
      htc_phone: HTC_ONE_A9
do my_phone := i_phone
   my_phone := samsung_phone
   my_phone := htc_phone
```

  suggests that these instances may <mark>*substitute*</mark> for each other.

- Intuitively, when expecting SMART_PHONE, we can substitute it by instances of any of its **descendant** classes.

  ∵ Descendants *accumulate code* from its ancestors and can thus *meet expectations* on their ancestors.

- Such <mark>*substitutability*</mark> can be reflected on contracts, where a

  <mark>*substitutable instance*</mark> will:

  ○ ***Not*** require more from clients for using the services.

  ○ ***Not*** ensure less to clients for using the services.

```
┌──────────────────────────┐         my_phone      ┌──────────────────────────┐
│     PHONE_USER           │─────────────────────▶ │     SMART_PHONE          │
├──────────────────────────┤                       ├──────────────────────────┤
│ my_phone: SMART_PHONE    │                       │ get_reminders: LIST[EVENT]│
│                          │                       │  require ??              │
└──────────────────────────┘                       │  ensure ??               │
                                                    └──────────────────────────┘
                                                               △
                                                               │
                                                               │
                                                               │
                                      ┌──────────────────────────┐
                                      │    IPHONE_6S_PLUS        │
                                      ├──────────────────────────┤
                                      │ get_reminders: LIST[EVENT]│
                                      │  require else ??         │
                                      │  ensure then ??          │
                                      └──────────────────────────┘
```

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e: Result | e happens today
end
```

```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e: Result | e happens today or tomorrow
end
```

Contracts in descendant class *IPHONE_6S_PLUS* are *not suitable*.
($battery\_level \geq 0.1 \Rightarrow battery\_level \geq 0.15$) is not a tautology.
e.g., A client able to get reminders on a *SMART_PHONE*, when battery
level is 12%, will fail to do so on an *IPHONE_6S_PLUS*.

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e:Result | e happens today
end
```

```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e:Result | e happens today or tomorrow
end
```

Contracts in descendant class *IPHONE_6S_PLUS* are *not suitable*.
 (*e* happens ty. or tw.) ⇒ (*e* happens ty.) not tautology.
 e.g., A client receiving today's reminders from *SMART_PHONE* are
 shocked by tomorrow-only reminders from *IPHONE_6S_PLUS*.

Inheritance and Contracts (2.4)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e:Result | e happens today
end
```

```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e:Result | e happens today between 9am and 5pm
end
```

Contracts in descendant class *IPHONE_6S_PLUS* are *suitable*.

- ***Require the same or less*** $\alpha \Rightarrow \gamma$
  Clients satisfying the precondition for *SMART_PHONE* are ***not*** shocked by not being to use the same feature for *IPHONE_6S_PLUS*.

Inheritance and Contracts (2.5)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e: Result | e happens today
end
```

```
class IPHONE_6S_PLUS
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.05 -- 5%
    ensure then
      δ: ∀e: Result | e happens today between 9am and 5pm
end
```

Contracts in descendant class *IPHONE_6S_PLUS* are *suitable*.

- **Ensure the same or more** $\delta \Rightarrow \beta$
  Clients benefiting from *SMART_PHONE* are **not** shocked by failing to gain at least those benefits from same feature in *IPHONE_6S_PLUS*.

Contract Redeclaration Rule (1)

- In the context of some feature in a descendant class:
  - Use `require else` to redeclare its precondition.
  - Use `ensure then` to redeclare its precondition.

- The resulting *runtime assertions checks* are:
  - `original_pre or else new_pre`

    ⇒ Clients **able to satisfy** *original_pre* will not be shocked.

    ∵ **true** ∨ *new_pre* ≡ **true**

    A *precondition violation* will **not** occur as long as clients are able to satisfy what is required from the ancestor classes.
  - `original_post and then new_post`

    ⇒ **Failing to gain** *original_post* will be reported as an issue.

    ∵ **false** ∧ *new_post* ≡ **false**

    A *postcondition violation* occurs (as expected) if clients do not receive at least those benefits promised from the ancestor classes.

Contract Redeclaration Rule (2.1)

```
class FOO
  f
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f require else new_pre
    do ...
    end
end
```

- Unspecified *original_pre* is as if declaring `require true`

$$\because \textbf{\textit{true}} \vee \textbf{\textit{new\_pre}} \equiv \textbf{\textit{true}}$$

```
class FOO
  f
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    do ...
    ensure then new_post
    end
end
```

- Unspecified *original_post* is as if declaring `ensure true`

$$\because \textbf{\textit{true}} \wedge \textbf{\textit{new\_post}} \equiv \textbf{\textit{new\_post}}$$

Contract Redeclaration Rule (2.2)

```
class FOO
  f require
      original_pre
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    do ...
    end
end
```

- Unspecified *new_pre* is as if declaring `require else false`

  $\because$ *original_pre* $\vee$ **false** $\equiv$ *original_pre*

```
class FOO
  f
    do ...
    ensure
      original_post
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    do ...
    end
end
```

- Unspecified *new_post* is as if declaring `ensure then true`

  $\because$ *original_post* $\wedge$ **true** $\equiv$ *original_post*

Invariant Accumulation

- Every class inherits *invariants* from all its ancestor classes.
- Since invariants are like postconditions of all features, they are "***conjoined***" to be checked at runtime.

```
class POLYGON
  vertices: ARRAY[POINT]
invariant
  vertices.count ≥ 3
end
```

```
class RECTANGLE
inherit POLYGON
invariant
  vertices.count = 4
end
```

- What is checked on a RECTANGLE instance at runtime:

$(vertices.count \geq 3) \wedge (vertices.count = 4) \equiv (vertices.count = 4)$

- Can PENTAGON be a descendant class of RECTANGLE?

$(vertices.count = 5) \wedge (vertices.count = 4) \equiv$ ***false***

Inheritance and Contracts (3)

```
class FOO
  f
    require
      original_pre
    ensure
      original_post
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    require else
      new_pre
    ensure then
      new_post
    end
end
```

(Static) *Design Time* :

- ○ $original\_pre \Rightarrow new\_pre$ should be proved as a tautology

- ○ $new\_post \Rightarrow original\_post$ should be proved as a tautology

(Dynamic) *Runtime* :

- ○ $original\_pre \lor new\_pre$ is checked

- ○ $original\_post \land new\_post$ is checked