ETロボコン2014 Embedded Technology Software Design Robot Contest

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

HSCI Professional Learning Course Series

ETロボコン2014 南関東地区大会モデリングスペシャルセミナー

ソフトウェア正当性 と モデリングの科学

2014年6月21日

Takanari Hashimoto

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

SEI Partner CMMI

ごあいさつ

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

プロフィール

橋本隆成

- Hashimoto Software Consulting International Inc.
- カーネギーメロン大学ソフトウエア工学研究所パートナー
- カーネギーメロン大学ソフトウエア工学研究所認定CMMIインストラクタ

経歴

- ソニー(株)
- (株)オージス総研
- 日本ヒューレット・パッカード
- 三菱スペースソフトエア(株)

HSCIメール／ホームページ

- URL：http://hsc-i.com/index.html
- メール：hashimoto@hsc-i.com

アジェンダ

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

- プロローグ～問題提起とETロボコン設計コンテスト
- Chapter1～ソフトウエア正当性と科学的モデリングアプローチ
- Chapter2～科学的継承設計理論コンメンタール
- Chapter3～DesignByContract™
- Chapter4～DesignByContract™によるタイプ(型)置換原則の実現
- Chapter5～タイプ(型)&モジュラー推論と安全な再利用
- WRAP-UP～設計コンテストにGO!!

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

プロローグ
《動機づけ編》
問題提起と
ETロボコン設計コンテスト

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

本日のテーマ

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

ETロボコンを通じて考えよう！

本日のテーマ：

“正しい”設計＆”良い”設計とは何か？

問題提起～ソフトウエアの品質と生産性

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

品質保証と生産性

• **欠陥(不具合)**による**手戻りコスト**は「開発総コスト」の**40%～60%**を占める
  • [C.Jones1986]/
  • [Coper and Mullen1993]/
  • [Haley1996]/
  • [Wheeler,Brykczynski and MeesonJr.1996]
• 下流工程に行くほど**欠陥摘出は困難**で修正コストが増大し**生産性が低下**する
• 「統合テスト」だけでは**全体のコードの20%**程度しか検証できない

問題提起～欠陥摘出率

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

項目	インスペクション	テスト
重大な欠陥摘出の比較 [Kaplan1995@IBM Santa Teresa研究所]	• コードインスペクションで平均**3.5人**	• テストフェーズの作業で平均**15～20人**
欠陥摘出率 [C.Jones]	• 設計／コードインスペクションで平均**50～70%**の摘出率 • 最高は約**95%**の除去率	• テスト工程で**35%以下**の除去率 • 特殊ツールを使わない限りの摘出率**60%**を超えられない
M.Johnson 調査1994	• インスペクション技法を用いると,どのコードを調べたか正確に把握できる	• コードカバレッジアナライザで測定しない限り,単体テストでテストできるコードは全体の**60%**を超えない

問題提起～欠陥摘出率と品質

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

ケース	効果説明
Russell1991@ Bell Northern Research [Holland1999]	• リアルタイムシステムで250万行のコードインスペクションを実施し，欠陥を1つ摘出する毎に保守工数を平均**33時間減少**させることに成功 • インスペクションの欠陥摘出はテストよりも**2～4倍速い**
Primark Investment Management Services	• 広範囲に体系的なインスペクション戦略を展開して最初の5年間で**総計約30,000人時の工数削減**に成功 • 「1顧客／1年」に報告される製品の欠陥数は**5分の1**に減少
Imperial Chemical Industries [Gilb and Graham1993]	• インスペクションを実施した約400本のプログラムの保守コストは,実施していない同様の400本のプログラムと比較して保守コストが**10分の1**と判明
Litton Data Systems [Madachy1995]	• プロジェクト**総工数のうち3%だけ**をインスペクションに充てた結果，結合テストとシステムテストによる摘出された欠陥数が**30%減少** • さらに設計およびコードのインスペクションによって製品統合の工数を**50%削減**した

問題提起～インスペクションのコスト削減効果

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

ケース	効果説明
Raytheon Electronic Systems [Haley1996]	• プロジェクトの総コストに占める手戻りを2年間で**41%から20%に減少** • 効果の主要因は**インスペクション**の導入 • 結合テストで摘出されるコード欠陥の修正コストを**80%減少** • 再テストのコストが**半分**
スペースシャトル開発 [Paulk1995]	• 開発上流工程のインスペクションで検出した１欠陥の修正コストを**1ドル**とすると 　• 「システム統合テスト」工程で検出し修正した場合は**13ドル** 　• 「出荷後」で検出した場合は**92ドル**
Hewlett-Packard [Grady and Van Slack1994]	• インスペクションの効果は,**ROIは10対1**で,**推定節約額は年間2,140万ドル**を達成 • 設計工程にインスペクションを適用し製品化の時間を**1.8か月短縮**
T&T Bell Laboratories [Humphrey1989]	• インスペタションで欠陥摘出コストが**10分の1に削減**した • インスペクションで品質は**10倍**改善され,生産性は**14%向上**した
Holland1999 @IBM	• インスペクションを1時間実施する経済効果は： (インスペクションで摘出出来たはずの)欠陥が製品に残ったままリリースされた場合のテストに掛かる工数**20時間**と手戻りに掛かる工数**82時間**が節約される

問題提起～インスペクション実施速度と量

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

研究報告	推奨値
Kerl.E.Wiegers	• ソースコード**150～200Loc/Hour** • ドキュメント**3～4ページ/Hour**
Gilb＆Graham	• ソースコード**50Loc/Hour** • ドキュメント**0.5～1ページ/Hour**
IBMの事例 [Frank Buck 1980]	• COBOLの最適速度は**95Loc/Hour** • この数値より**35%**程度多い**135Loc/Hour**にすると欠陥摘出数が**30%も低下**する

問題提起～興味深いインスペクションの事実

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

修正ミスの起こる割合[Fagan1986]
• **6回に1回の割合**で欠陥修正に誤りや不完全な修正がある • (＊)この誤りや不完全な修正数には,修正作業で新たに摘出された欠陥数を含む

インスペクションの効率[IBM Rochester研究所1992]
• インスペクションでも一度の実施で**12%～40%**は欠陥摘出できない

インスペクションミーティングの効率[Votta1993]
• ミーティング時の欠陥摘出は**全欠陥の4%**に過ぎない報告がある

問題１～品質判断の問題

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

品質が良いか？悪いか？[ハンフリー@カーネギーメロン大学]

- 欠陥数100件／100万LOC ☞ 品質は良いか？悪いか？
- 欠陥数10件／100万LOC ☞ 品質は良いか？悪いか？
- 欠陥数1件／100万LOC ☞ 品質は良いか？悪いか？
- 欠陥数100件／100万LOCの航空機や自動車 ☞ 安全か？乗りたいか？

問題１の考え方～品質の視点と定量化

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

品質の定量化[ハンフリー@カーネギーメロン大学]

- CMMレベル1の組織：
  - 欠陥5件／1000LOC
- CMMレベル5の組織：
  - 数百万LOCの規模になると出荷した未発見欠陥が1000個／100万LOC
- 1件／1000LOCの欠陥が存在する場合：
  - 欠陥1000件／100万LOC(1件／リスト30ページ)
  - 欠陥100件／100万LOC(1件／リスト300ページ)
  - 欠陥10件／100万LOC(1件／リスト3000ページ)

- (参考データー)：
  - W.ハンフリー作成のC++プログラムが996LOCでリスト30ページ

問題２～問題１に加え下記について考える

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

高品質＆高生産性は何できまるのか？

?

- 高品質のモデルとは？
- 高品質のプログラムとは？

欠陥(不具合)がないことをどの様に判定・証明するのか？

?

- モデルやプログラムに**欠陥が無い事**をどの様に客観的に示すのか？

品質目標と開発戦略の不在

- 多くの組織において品質目標が無い
- 品質や生産性のデーターを持っていない
- 開発戦略や品質戦略が無い

教訓～ソフトウエア再利用の大参事

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

非科学的な再利用や差分開発のアプローチは悲惨な結果招く

- ソフトウエアコンポーネントの**再利用が原因の大参事**として**Ariane5**がある
- 原因調査結果：
  - **例外がキャッチ出来無い不具合**が未修正の**Ariane4**のソフトウェアコンポーネントの再利用が原因で宇宙船のソフトウエアが破壊された[Lio96]
  - JzquelとMayerは,事故の根本原因は**コンポーネントの例外の「契約」が存在しない**事と主張[JM97]
- JzquelとMayerは結論付けている:
  - **「契約」を定義せず「タイプ(型)置換原理」を満足しない再利用**は**完全な愚行**である

ビジョンの設定

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

テーマ:“正しい”設計＆”良い”設計とは何か？

ETロボコンを通じて考えよう！

- **科学的なモデル&プログラムについて理解する**
  - **設計の正当性(correctness)がある／判断できる**
  - **安全な再利用が可能**
  - **インスペクションやテスト工数が削減**
  - **高品質＆高生産性を担保**
  - **客観的なデーターによる品質評価**
  - **自動化が可能**

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

Chapter1
《準備編》

ソフトウエア正当性と科学的モデリングアプローチ

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

プログラムの正当性(correctness)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

プログラムの正当性(correctness)

- **プログラムの正当性：**
  - Hoareトリプル：**{P} C {Q}**
  - インスペクションやテストをしなくても**「正しさ(correctness)」を保証**
  - **演繹的アプローチ**

モデルの正当性(correctness)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

モデルの正当性(correctness)

• Hoareトリプルの理論は分析・設計にも応用されている
  • **Design By Contract™**
  • **不変条件(invariant)**
  • **事前条件(pre-condition)**
  • **事後条件(post-condition)**
  • **タイプ(型)置換原理**

正当性(correctness)と科学的モデリング

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

正当性(correctness)と科学的モデリング

- **ミッション・クリティカルなシステム**には下記が重要となる
  - **タイプ(型)安全の利用：**
    - タイプ(型)安全を考慮した設計手法
    - 強い型づけプログラミング言語の利用
    - タイプ(型)置換を保証する再利用
    - タイプ(型)置換の手法
  - **演繹的アプローチ：**
    - 数学的／論理学的な定理・原理
    - タイプ(型)置換原理(リスコフ置換原理など)
    - タイプ(型)推論
    - モジュラー推論
    - etc
  - **演繹的アプローチを支援する原則・法則に準拠：**
    - 開放閉鎖原則
    - デミテルの法則

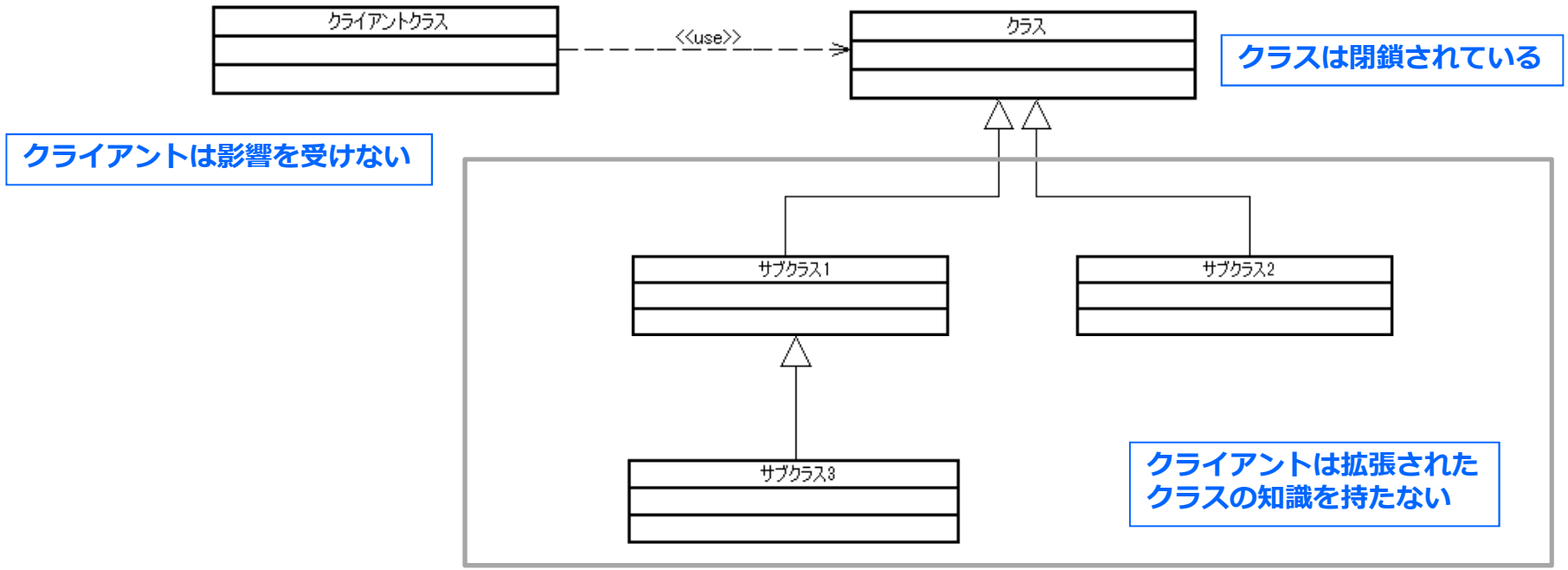
(＊)本セミナーはビジネスのためのソフトウエア開発を対象とする

開放閉鎖原則(Open-Close Principle)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

開放閉鎖原則(Open-Close Principle)[Meyer86]

- クラス／コンポーネント／モジュールが：
  - **拡張可能**で**クラス／コンポーネント／モジュールは開放**されていること
  - クライアントに対して**クラス／コンポーネント／モジュールは閉鎖**されていること

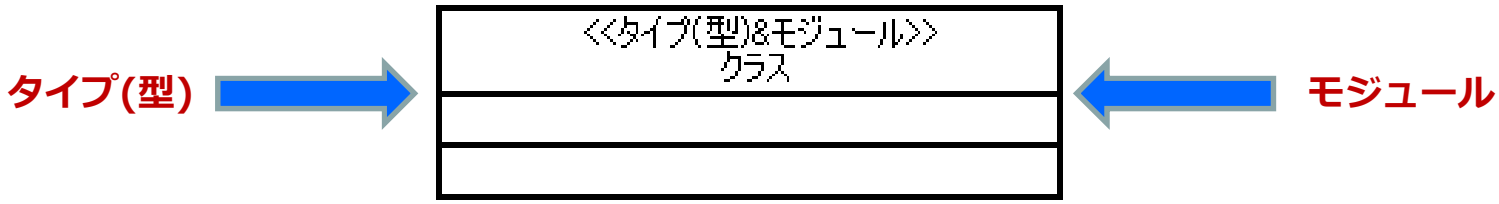


抽象データー型のタイプ(型)とモジュール

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

抽象データー型をタイプ(型)として見るか？モジュールとしてみるか？

- **抽象データー型(クラスetc)**は**タイプ(型)**の性質と**モジュール(プログラム)**の性質を併せ持つ
- タイプ(型)とモジュール(プログラム)が**一致するとき最大の相乗効果を発揮**する
- 注意しなければならない**パラドックス**も持ち合わせる



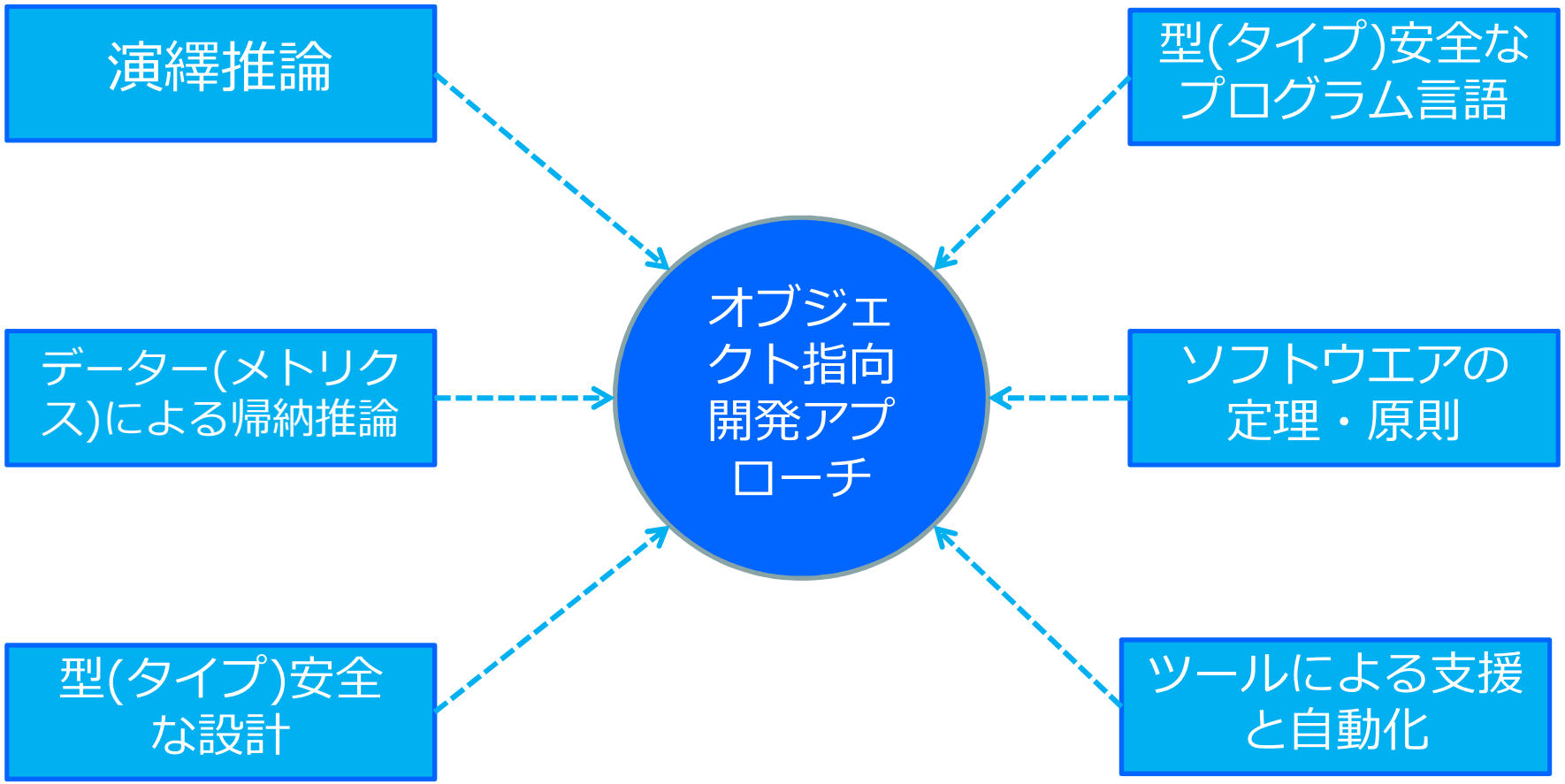
特性	タイプ(型)	モジュール(プログラム)
特性(属性や操作)の追加	○	○
再定義(オーバーライド)	○	○
情報隠蔽	○	○
継承	○	○
多相(ポリモフィズム)	○	×
動的束縛	○	×

タイプは全ての特性を持つ

(＊)但し研究者によって異なることがある

先進オブジェクト指向開発

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.



(＊)ビジネスのためのソフトウエア開発を対象とする

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

Chapter2
入門編
科学的継承設計理論コンメンタール

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

科学的な継承理論の理解が鍵

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

正しい継承理論を知ろう
• **正しい設計には,継承を正確に使用する事が極めて重要**である[Ian Joyner] • **継承により開放閉鎖原則(Open-Close Principle)を最大限に活用**できる[Meyer] • **OO開発の威力の多くは継承の重要な概念によるもの**である[Herbert Toth]

非科学的な継承の理解

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

従来の継承の理解～継承のメリットを活かせず不具合を誘発する

- 継承関係は「is-a」関係,「kind-of」関係で判定する
- 「実装継承」は間違いである
- Javaのインタフェース(Interface)を利用した継承をなるべく利用すべき
- 多重継承は避けた方が良い
- Etc.

継承のメリットを活かせず不具合を誘発する理由

- 継承関係は「is-a」関係,「kind-of」関係は継承関係の判定の基準にならない
  - 継承の設計の基準に役に立たない
- 全ての“モノ”に「is-a」関係が存在する
  - 「**醜い家鴨の子定理(ugly duckling theorem)**」[渡辺1978]
- Javaのインタフェースは「振る舞いサブタイプ(型)継承」ができない
- 多重継承&実装継承による**ミキシン(Mix-in)クラス**は非常に便利

理由

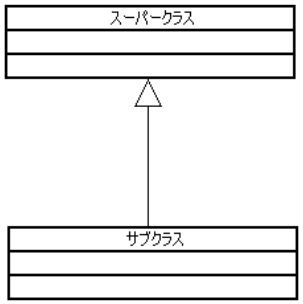
科学的な継承理論の理解

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

科学的な継承理論の理解のポイント

- 継承を下記の視点から理解する
  - **実装継承／タイプ(型)継承**
  - **タイプ(型)の拡張／制限／特殊化**
  - **タイプ(型)置換原理(substitute principle)**
  - **タイプ(型)推論／モジュラー推論**
  - **例外メカニズム**
  - **プログラム言語仕様**
  - **コンパイル&リンク**

- 上記の視点から継承および継承階層を考えることで,**不具合が無い**,**首尾一貫**した**工学的な継承の設計**が可能となる
- 逆に言えば,上記の視点を欠いた継承の設計は,**不具合を誘発する極めて危険な設計アプローチ**である

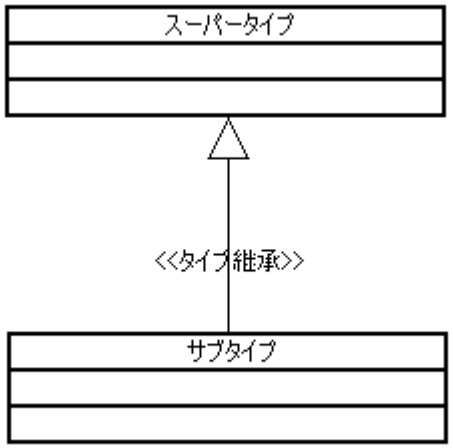


実装継承 vs. タイプ(型)継承

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

継承の基礎～継承は「実装継承」と「タイプ(型)継承」に大別できる[Bla91]

- **実装継承(implementation inheritance)**
  - スーパークラスの**実装**を再利用する
  - スーパークラスのから振る舞い(意味)の継承は必ずしも無い
  - 多重継承を利用したミキシン(Mix-in)クラスが代表例
- **タイプ(型)継承(type inheritance)**
  - スーパークラスの振る舞い(意味)を継承する
  - スーパークラスとサブクラスの置換を**意味**の視点から可能とする

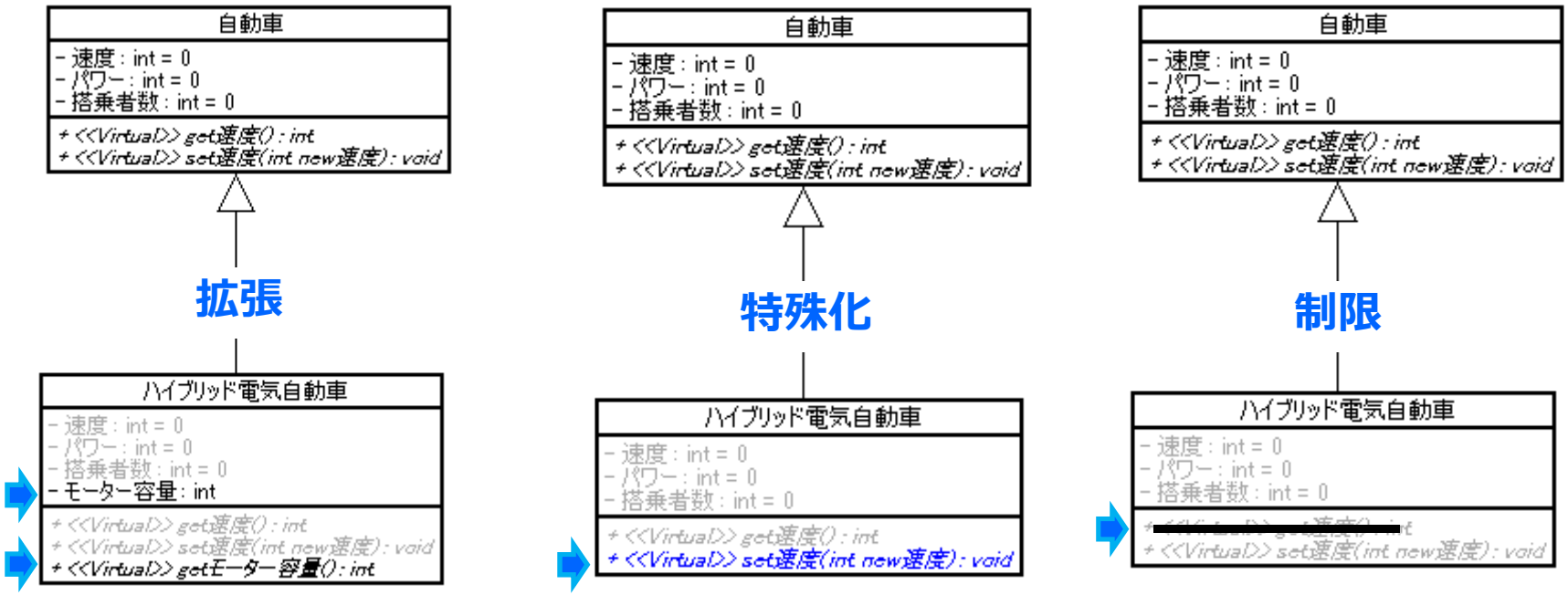


タイプ(型)継承の分類～拡張／制限／特殊化

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

タイプ(型)継承によりサブタイプは下記のケースが発生する[Bla91]

- **タイプ(型)拡張(extension)**
  - サブクラスは新しい属性・操作を追加すること
- **タイプ(型)特殊化(specialization)/再定義(override,redefine)**
  - サブクラスで最適化するために操作を再定義して特殊化すること
- **タイプ(型)制限(restriction)**
  - スーパークラスの属性や操作の一部を継承すること

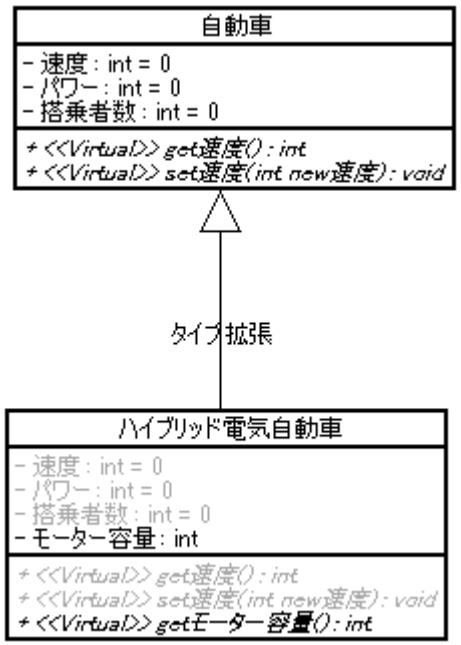


タイプ(型)継承とタイプ(型)関係

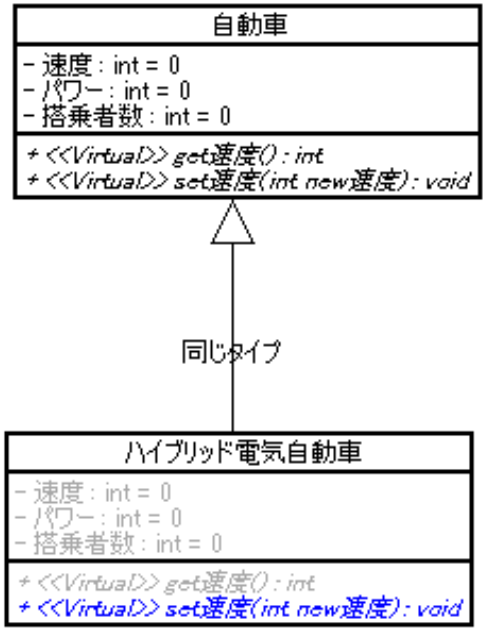
HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

タイプ(型)継承の分類	クラスから見た関係	タイプ(型)から見た関係
拡張	スーパークラス-サブクラス関係	スーパータイプ-サブタイプ関係
特殊化/再定義	スーパークラス-サブクラス関係	同じタイプ同士関係
制限(制約)	スーパークラス-サブクラス関係	サブタイプ-スーパータイプ関係

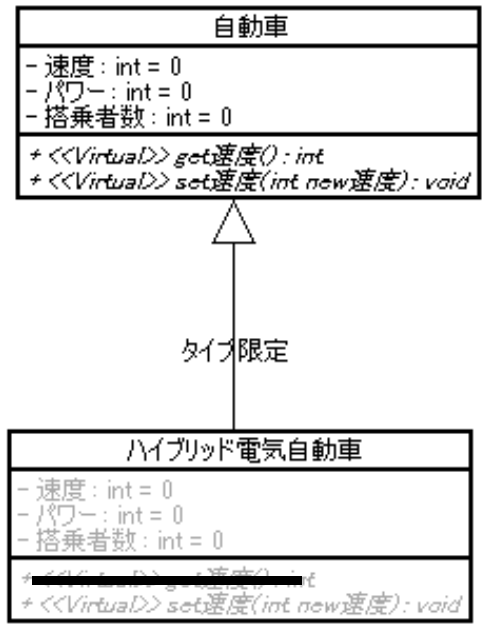
<スーパータイプ-サブタイプ>関係



<同じタイプ同士>関係



<サブタイプ-スーパータイプ>関係



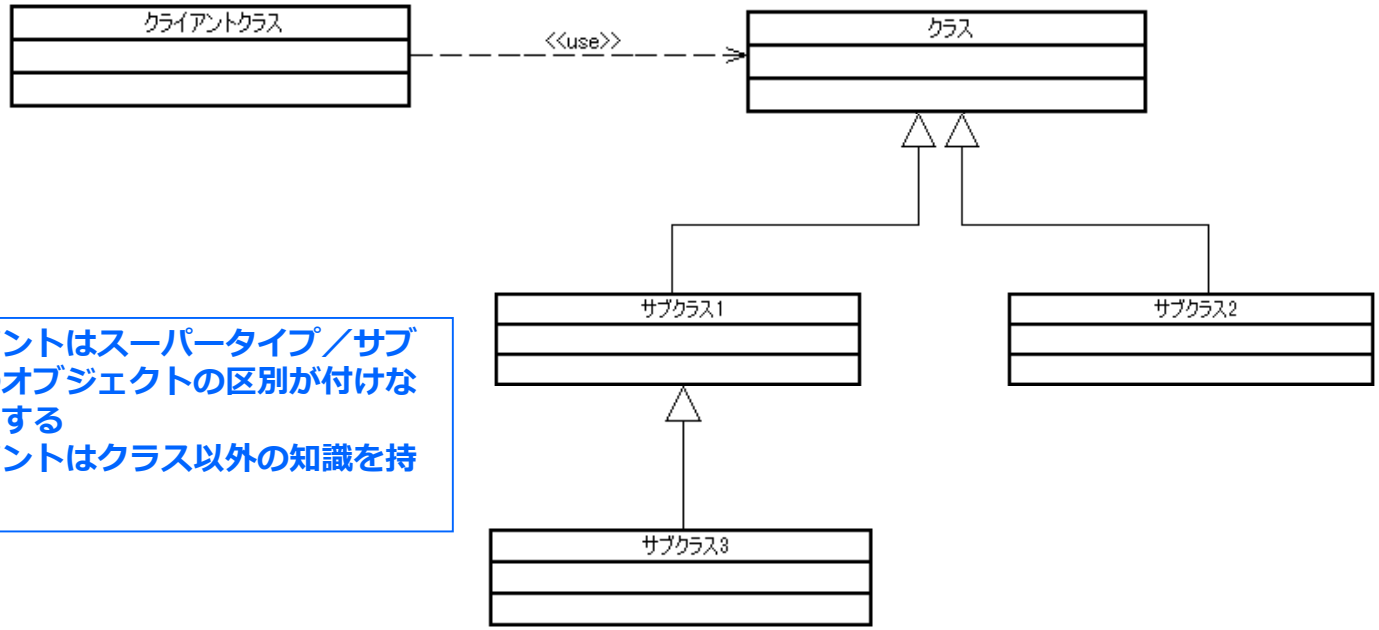
タイプ(型)置換原理(type substitution principle/type conformance principle)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

最重要基礎事項～タイプ(型)とクラスの区別とタイプ(型)置換原理

- **サブクラス(subclass)**と**サブタイプ(subtype)**は明確に異なる概念である
- 継承階層内でサブクラスは任意のスーパークラスと**タイプ(型)置換原理**を満たす必要がある

サブタイプのオブジェクトをスーパータイプのオブジェクトとして**「意味を変えずに代わりに利用できる」**こと



- **クライアントはスーパータイプ／サブタイプのオブジェクトの区別が付けないで利用する**
- **クライアントはクラス以外の知識を持たない**

タイプ(型)置換原理を満たす

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

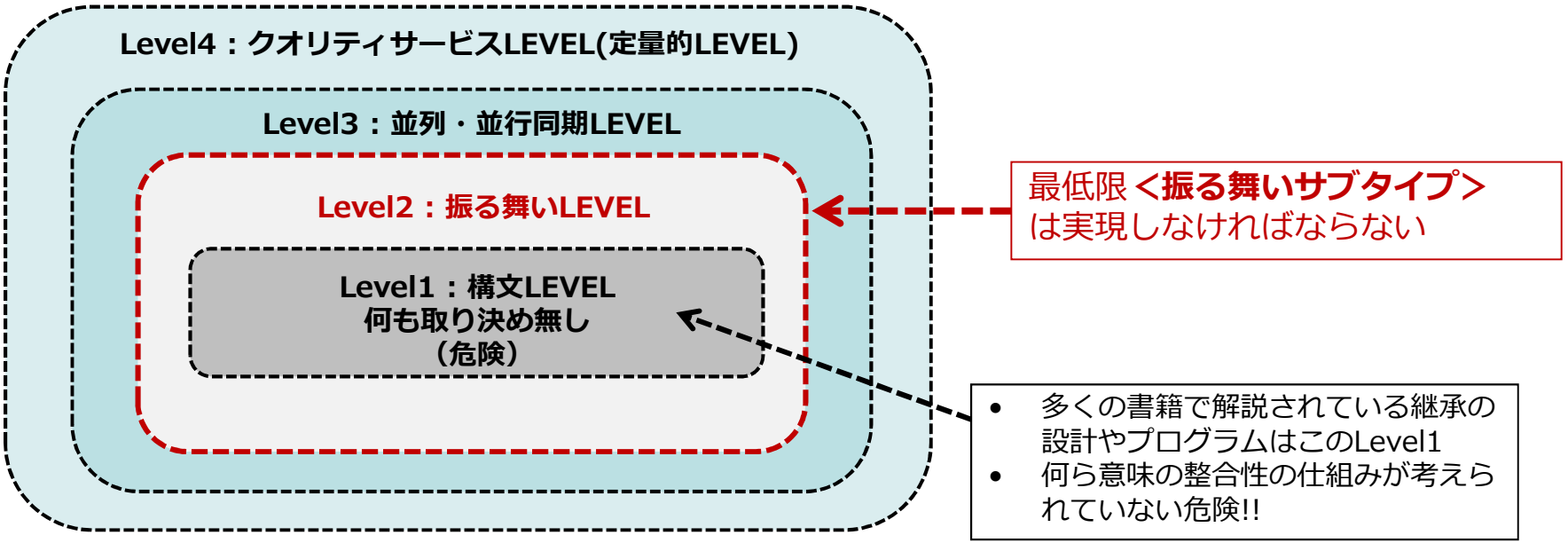
タイプ(型)置換原理を満足するとはどういう事か？
• **タイプ(型)置換Level**の**Level２以上の置換を実現する** • 継承されるサブタイプの**特性の全てが置換原理を満足すること** • タイプ(クラス)に**契約(表明)**で置換原理を満足する • **リスコフ置換原則**を満たす • タイプ(型)置換原則の判定に**論理式**を用いる

タイプ(型)置換の難易度Level

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

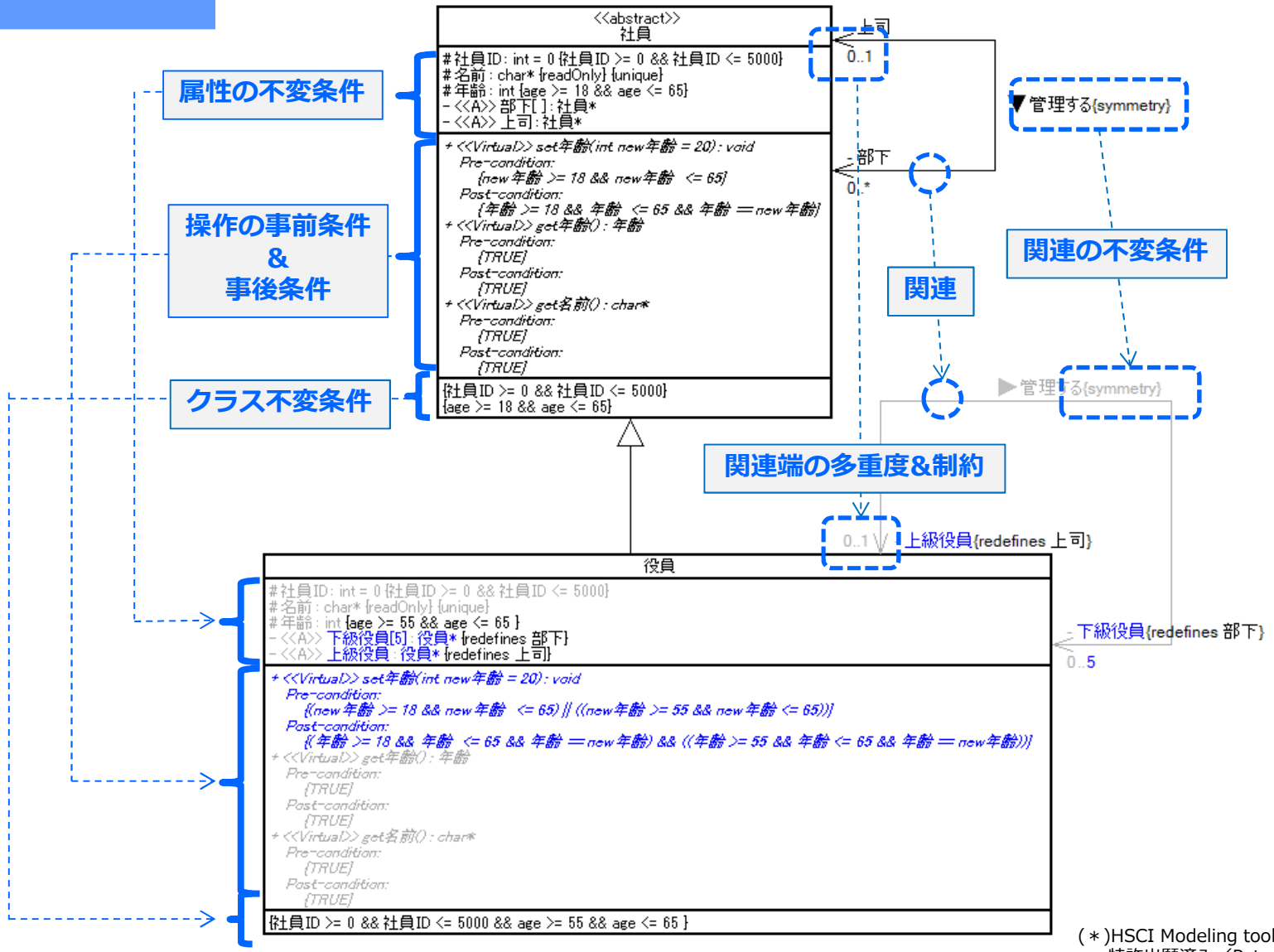
タイプ(型)置換のLevelがある[Beugnard1999]

- Level1：構文的置換(syntactic proof)：構文
- Level2：**振る舞い置換(behavioral proof)**：意味
- Level3：**処理順序置換(sequencing proof)**：並列・並行性
- Level4：**サービス品質置換(quality of service proof)**：処理時間,サイズ空間,安全性,etc



タイプ(型)置換原理の対象となるプロパティ(特性)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



(＊)HSCI Modeling tool
特許出願済み／Patent Pending

リスコフ置換原理
(LSP：Liskov Substitute Principle)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

リスコフ置換原理[Liskov et al.,1994]の意味

- スーパータイプとサブタイプにおいて下記を満たさなければならない：
  - スーパータイプの**不変条件(invariant)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない
  - スーパータイプの**事前条件(pre-condition)**はサブタイプにおいて＜弱める＞ことはできるが＜強める＞ことはできない
  - スーパータイプの**事後条件(post-condition)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことができない
- 「リスコフの置換原則」を満たす継承階層は,**継承階層にクラスを修正・追加・削除した後であっても決して意味的に破壊される事は無い**
  - **これによりスーパークラスとサブクラスの「意味(処理)の整合性を保証」した「真の多相(ポリモフィズム)」を実現でる**
  - **新規開発だけでなく,差分開発,再利用にいても最重要となる**

再定義(override/redefine)とタイプ(型)置換原理

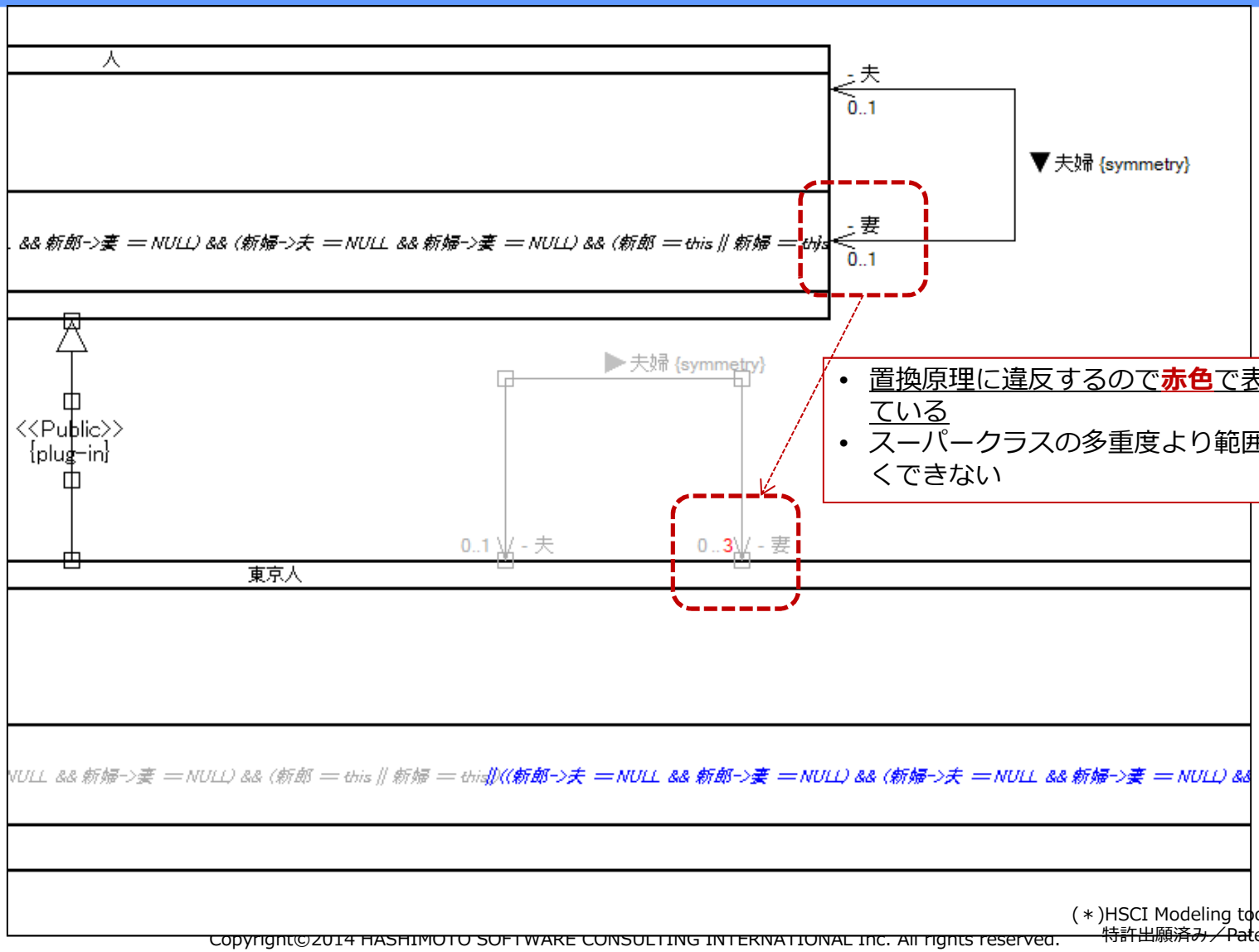
HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

タイプ(型)置換原理を満たすサブタイプの再定義(override/redefine)
• スーパータイプの特性の**「意味を変えず」**にサブタイプで**再定義(override/redefine)**する必要がある

特性(プロパティ)	サブタイプでの再定義	備考
関連名	再定義可能	• 但し研究者によって意見が異なることがある • プログラム言語によっては不可能な再定義(override/redefine)がある C++やJavaではできない
関連の参照方向	再定義不可能	
多重度	多重度の範囲を**狭くできる**	
関連の制約	**強める(増やす)**場合のみ再定義可能	
関連端(ロール名)	再定義可能	
関連端(ロール名)の可視性	再定義不可能	
関連端(ロール名)に付く制約	**強める**場合のみ再定義可能	

再定義(override/redefine)とタイプ(型)置換原理の判定の例

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



- 置換原理に違反するので**赤色**で表示されている
- スーパークラスの多重度より範囲を大きくできない

(＊)HSCI Modeling tool

モジュラー推論(Modular Reasoning)の規則

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

推論の論理式
• リスコフ置換原理に準拠した継承階層では**推論が可能**となる • **「コンポーネント検索」「コンポーネントの再利用」「(継承階層)振る舞いサブタイプ置換」**の推論 • ３つの視点からは論理式を使う前提と条件が異なるので注意

厳格	Specification Match名称	Specification Match論理式
1	exact pre/post[Zaremski97][Chen00]	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
2	exact pre	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
2	exact post	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
4	plug-in[Zaremski97]	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
5	weak-plug-in([Penix99]) *[Zaremski97]ではguarded plug-inと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{pre} \wedge SC_{post} \Rightarrow C_{post})$
6	satisfies[Penix99] *[Chen00]ではrelaxed plug-in **[Fischer97]ではplug-in compatibilityと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (C_{pre} \wedge SC_{post} \Rightarrow C_{post})$

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

Chapter3
《基礎編》
DesignByContract™

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

Design By Contract™(DbC:契約による設計)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

Design By Contract™(契約による設計)[Bertrand Meyer]

- Design By Contract™(契約による設計)を実践することで:
  - **リスコフ置換原理**の考えを実装できる
  - **Level２以上のタイプ(型)置換原則(LSP含む)を満すクラス階層が設計可能**
  - **タイプ(型)安全なシステムを効率的かつ高品質で開発可能**
  - **タイプ(型)安全なソフトウエアの再利用／差分開発**
  - **レビューやテスト工数の削減**
- Design By Contract™(契約による設計)ではタイプに**契約(表明)**を定義する
  - **クラス不変条件(タイプ不変条件)**
  - **属性の不変条件**
  - **操作の事前条件**
  - **操作の事後条件**

Design By Contract™(DbC:契約による設計)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

クラス不変条件・事前条件・事後条件はタイプ(型)の一部である

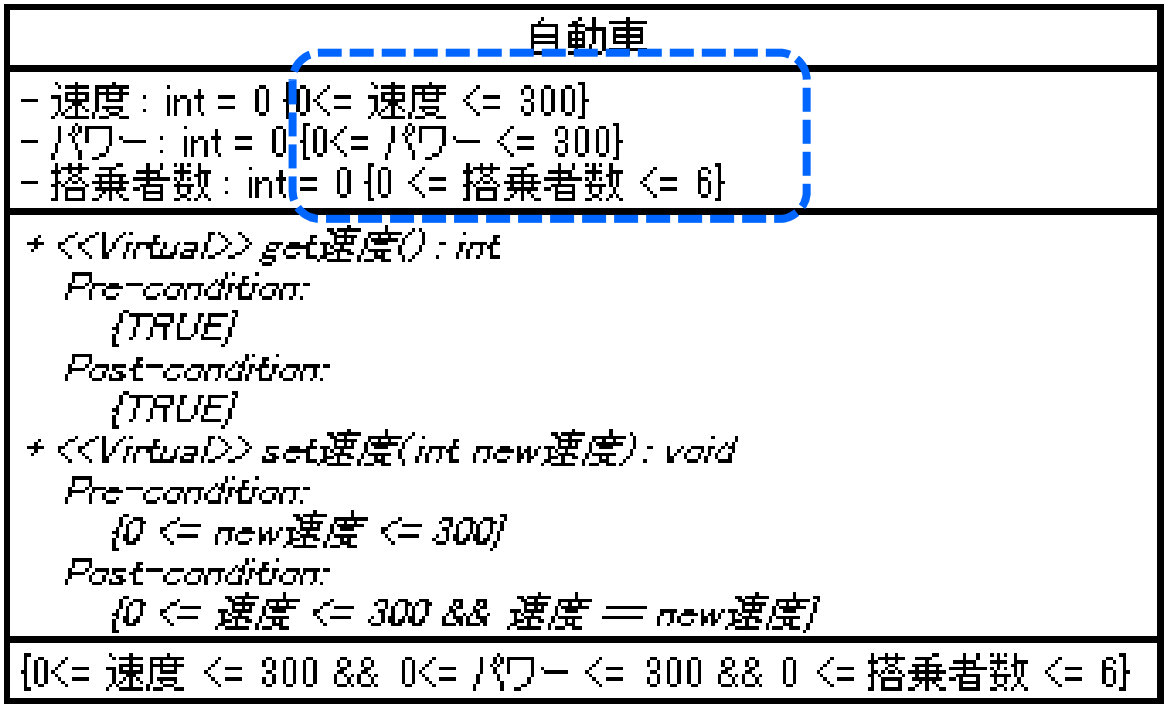
契約に基づく設計(Design By Contract)	
クラス不変条件 (class invariant)	• クラス全体の不変条件 • クラスのインスタンスが満たさないといけない命題を表現している
属性の不変条件 (invariant)	• クラスの属性が常に満たさなければならない条件
事前条件 (pre-condition)	• 操作を実行する前に成立しなければならない条件 (操作が呼び出された直後に判定される) • 事前条件を成立させる責任は操作を呼び出すクライアント側となる
事後条件 (post-condition)	• 操作が終了時に成立しなければならない条件(操作の最後で判定される) • 事後条件を成立させる責任は操作側となる

属性の不変条件(invariant)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

属性の不変条件(invariant)

- 属性の制約



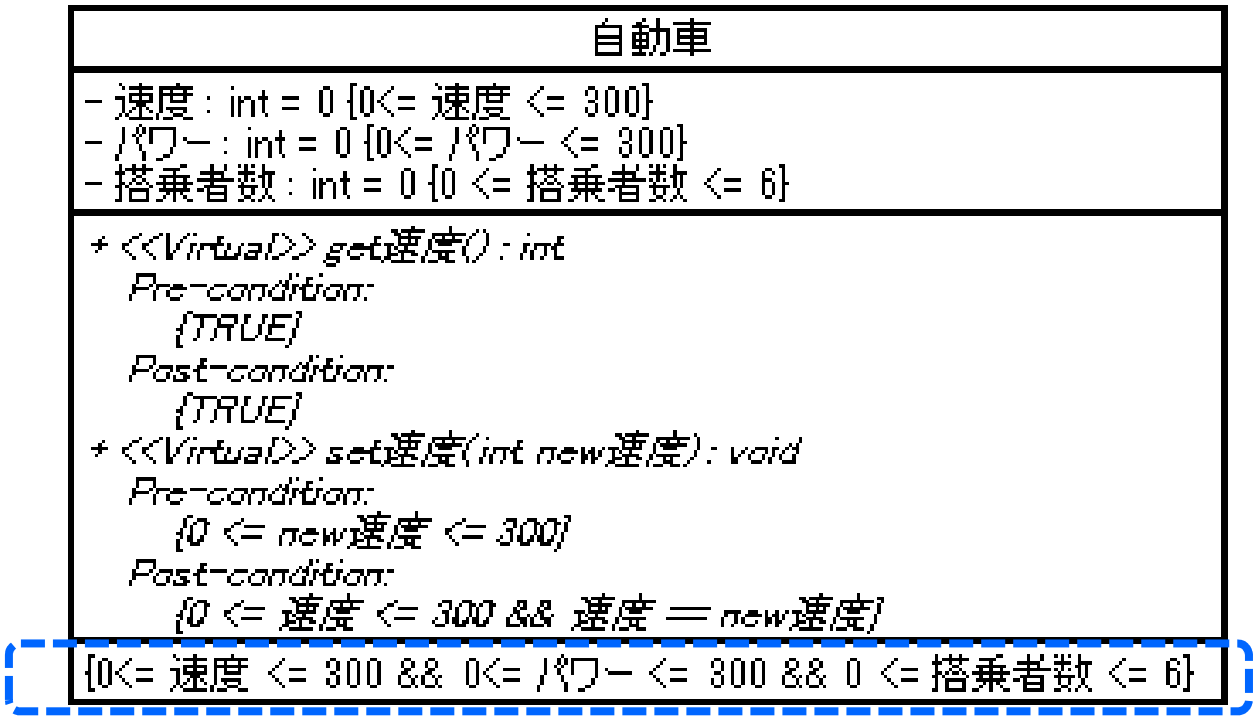
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

クラス不変条件(class invariant)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

クラス不変条件(class invariant)

- 属性の不変条件の総和
- 属性間の不変条件
- クラス不変条件が定義されない場合：
  - この時の条件は{**true**}



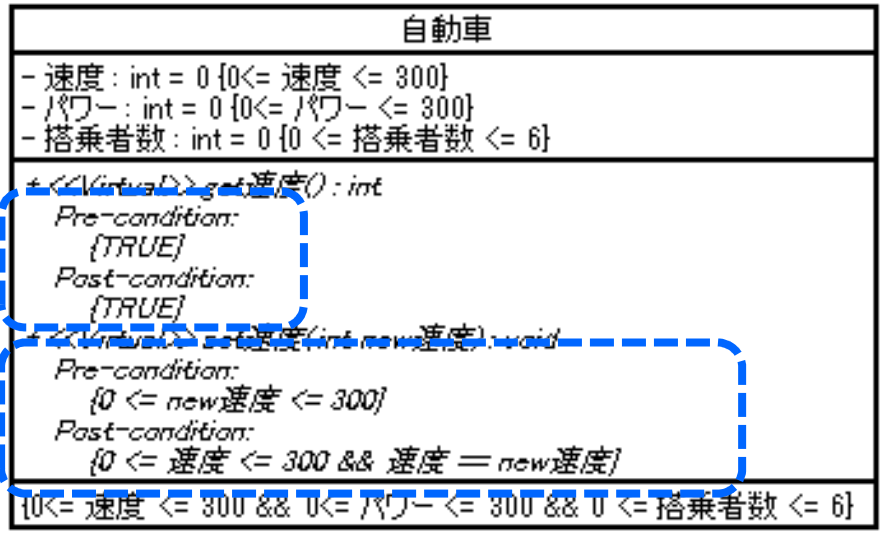
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

事前条件(pre-condition) & 事後条件(post-condition)①

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

事前条件(pre-condition)と事後条件(post-condition)

- 各操作に**事前条件(pre-condition)**と**事後条件(post-condition)**を定義
- **事前条件**：
  - 操作が意味をもつ条件を記載する
  - 開始条件
- **事後条件**：
  - 操作が保証する処理を記載する
- 事前条件(pre-condition)と事後条件(post-condition)が定義されない場合：
  - この時の条件は{**TRUE**}



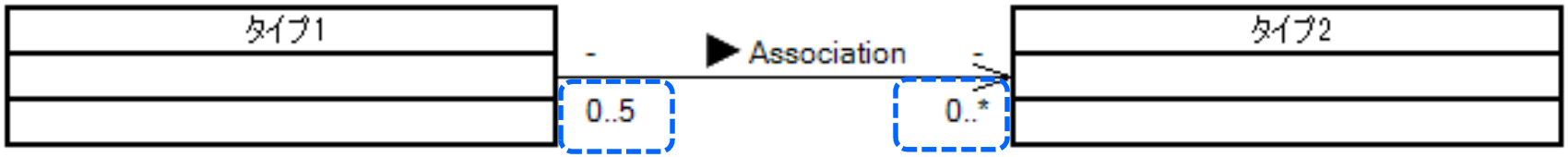
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

事前条件(pre-condition) & 事後条件(post-condition)②

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

事前条件(pre-condition)と事後条件(post-condition)

- 関連は**定義域**と**値域**が存在する：
  - 数学の写像／関数
  - 論理学の述語
- **多重度の[0]**があることは：
  - **部分関数(partial function)**を意味する
  - 事前条件を持つ事を意味する
- 事前条件は操作の**特性方程式(characteristic equation)**
  - その操作の**定義域(domain)**を決定する



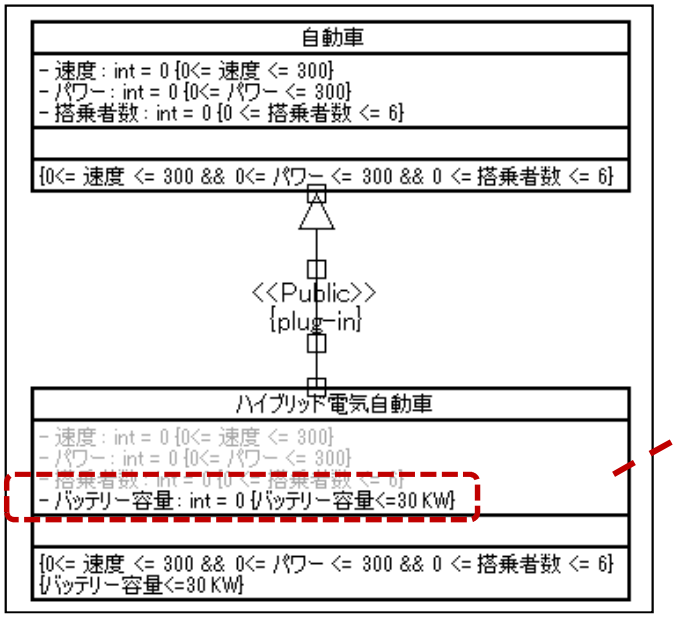
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

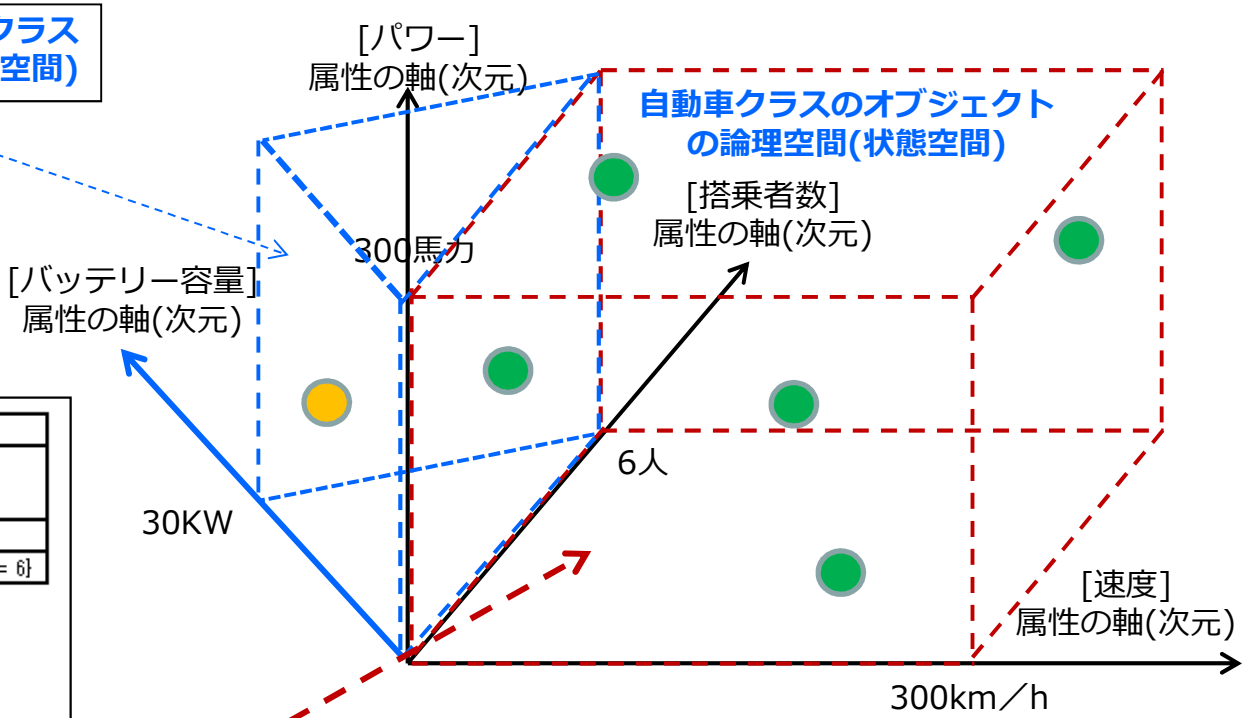
図解：Design By Contract™と振る舞いタイプ(型)置換①

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

拡張されたハイブリッド自動車クラスのオブジェクトの論理空間(状態空間)

タイプ(型)拡張
(サブタイプ電気自動車の固有の属性)



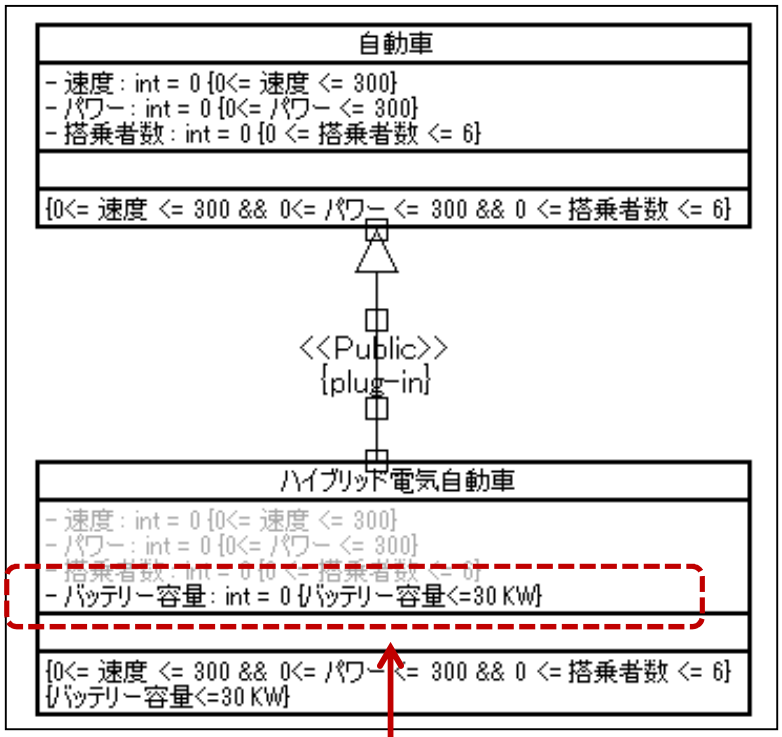


- **多相(ポリモフィズム)を使い正しい動作を保証するという事は,拡張されている軸は省くサブクラスの論理空間内は,スーパークラスの論理空間内と「等しい」か「内側」でなければならないことを意味する**
- **これが置換原理のまず最初の重要な意味である**

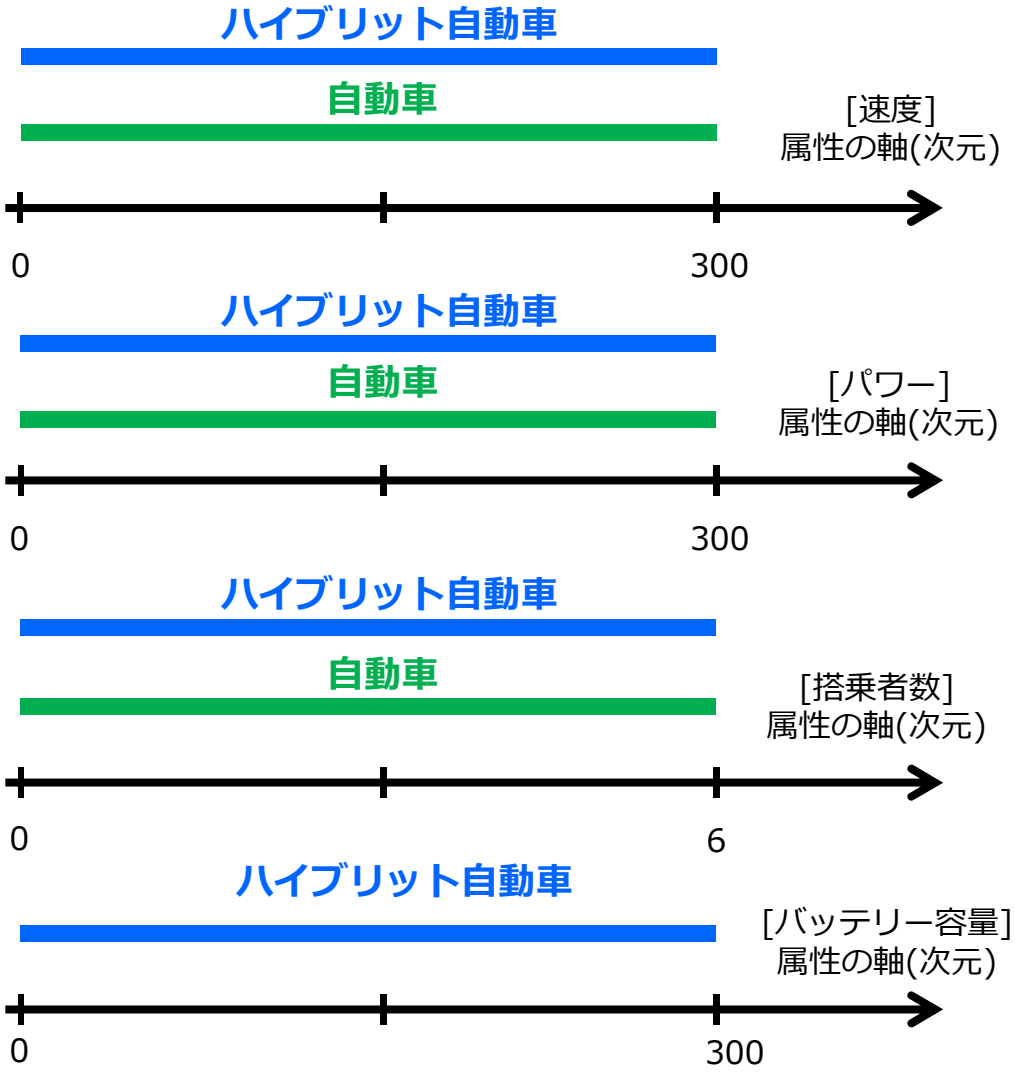
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

図解：Design By Contract™と振る舞いタイプ(型)置換②

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



サブタイプで「拡張」した部分は
スーパークラスとの置換原理の影響を
受けない



(＊)HSCI Modeling tool
特許出願済み／Patent Pending

リスコフ置換原理と論理空間(状態空間)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

リスコフ置換原理

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて**＜強める＞**ことはできるが**＜弱める＞**ことはできない
- スーパータイプの**事前条件(pre-condition)**はサブタイプにおいて＜弱める＞ことはできるが＜強める＞ことはできない
- スーパータイプの**事後条件(post-condition)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない

スーパータイプとサブタイプの論理空間(状態空間)の空間の関係

- 同じ次元サブタイプはスーパータイプの**「論理空間(状態空間)」**に**<完全一致する＞**or**<含まれる＞**必要がある
- サブタイプで**「拡張」**した部分は**置換原理の影響を受けない**
  - ただし,サブタイプで追加した属性はスーパータイプの属性に**「直交」**しなければならない

Design By Contract™(契約による設計)

- サブタイプの**属性の不変条件**／**クラス不変条件**は,スーパータイプより**<強くする>**ことは可能

HSCI Professional Learning Course Series

Chapter4
《実践編》

DesignByContract™ による タイプ(型)置換原則の実現

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

リスコフ置換原理の論理的解釈

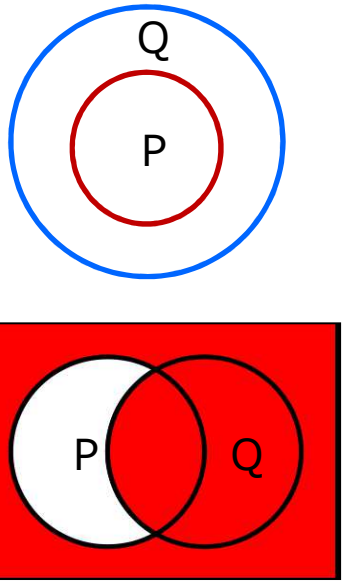
HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

リスコフ置換原理[Liskov et al.,1994]の意味

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない
  - **属性不変条件**
  - **クラス不変条件**
  - **$SC_{invariant} \Rightarrow C_{invarinat}$**
- スーパータイプの**事前条件(pre-condition)**はサブタイプにおいて＜弱める＞ことはできるが＜強める＞ことはできない
  - **操作(メソッド／関数)の事前条件**
  - **$C_{pre} \Rightarrow SC_{pre}$**
- スーパータイプの**事後条件(post-condition)**はサブタイプにおいて＜強める＞ことはできるが＜弱める＞ことはできない
  - **操作(メソッド／関数)の事後条件**
  - **$SC_{post} \Rightarrow C_{post}$**

含意(implies／⊃／⇒)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

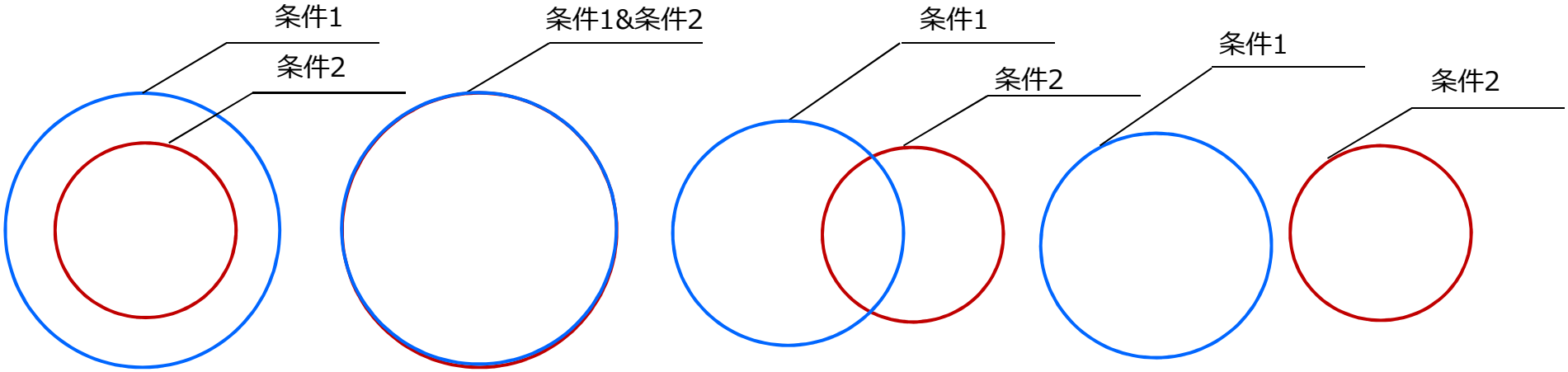
含意(imples／⊃／⇒)		
演算名称	• **含意(implies)**	
読み方	• PならばQ • PはQを**含意する** • PはQの**十分条件**(QはPの**必要条件**)	
記号	• P**ならば**Q • P⊃Q • P⇒Q (＊)集合の記号(⊃)とは別物	
論理式	• P⇒Q ≡ ¬P ∨ Q ≡ ¬(P∧¬Q)	
PならばQ	• QはPの必要条件(Necessary Condition) • PはQの十分条件(Sufficient Condition)」 • P≡Qのとき必要十分条件(if and only if ：iff：⇔)」	

強い(厳しい)と弱い(緩い)

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

含意(implies：十分条件)と「強い」「弱い」「同じ」

- **スーパータイプとサブタイプでは含意(implies：十分条件)の関係**
- 含意関係で「強い」「弱い」「同じ」は判定可能
- ２つの条件が＜互いに素＞＜交差する＞ときは「強い」「弱い」「同じ」は判定できない
- 比較不可な例：
  - 条件1[-10 < x < 5]
  - 条件2[-20 < x < 3]

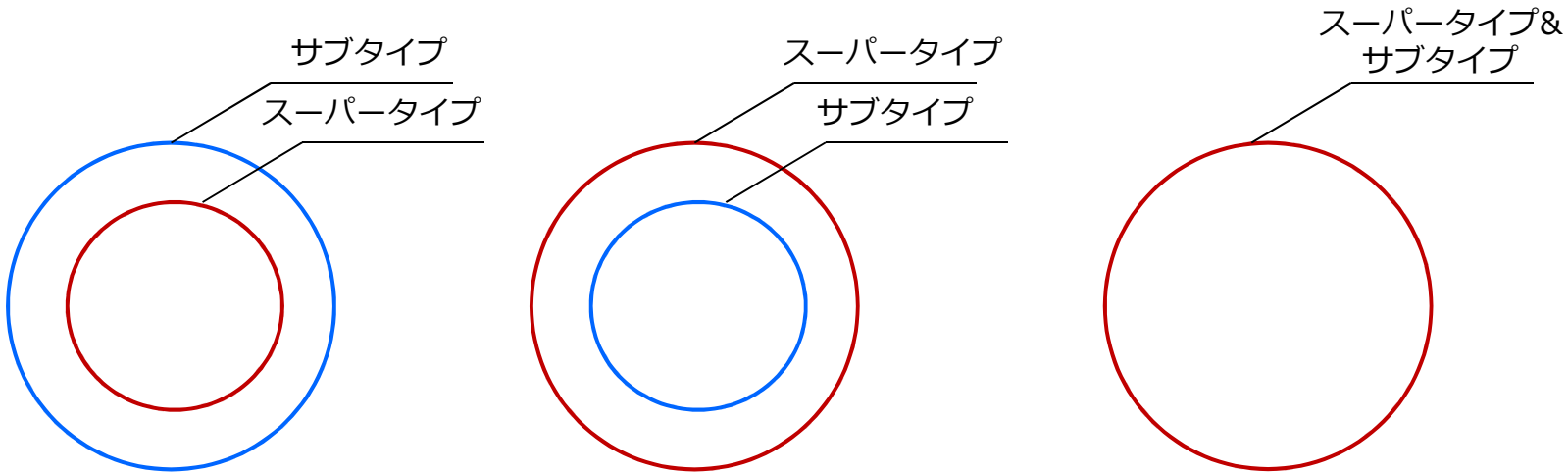


スーパータイプとサブタイプの条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

スーパータイプとサブタイプの条件([plug-in]のケース)

- タイプ(型)置換原則は満たす条件(**[plug-in]**のケース):
  - **クラス不変条件　　: $SC_{invariant} \Rightarrow C_{invarinat}$**
  - **操作事前条件　　: $C_{pre} \Rightarrow SC_{pre}$**
  - **操作事後条件　　: $SC_{post} \Rightarrow C_{post}$**
- **条件の論理的包含関係**を持つ継承階層を構築するこが**タイプ置換原則を実現すること**である



- **事前条件**

- **クラス不変条件**
- **事後条件**

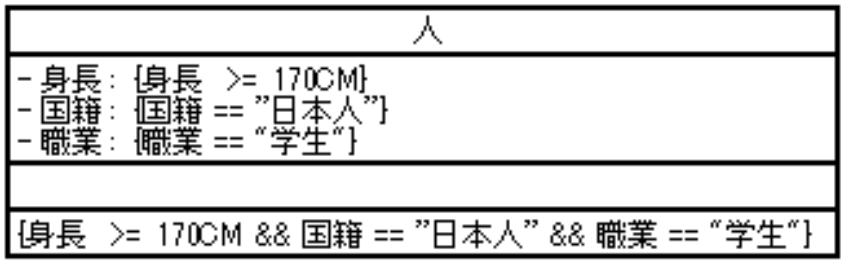
- **クラス不変条件**
- **事前条件**
- **事後条件**

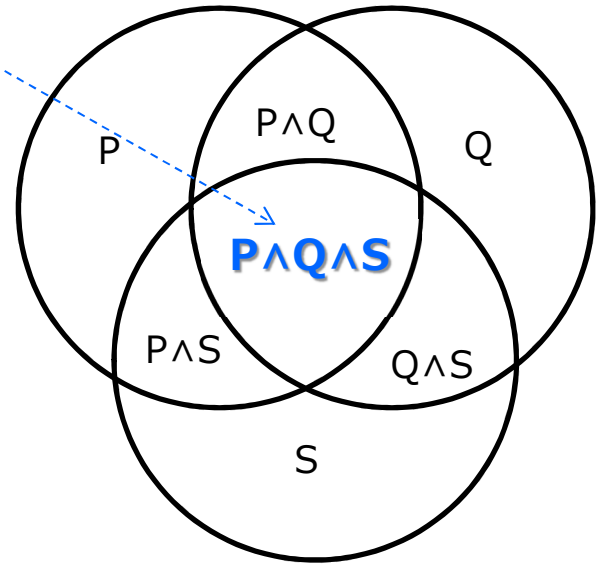
属性とクラス不変条件と論理積

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

属性の不変条件とクラス不変条件

- 3つの不変条件P,Q,Sがあるとき：
  - P(国籍)：日本人である
  - Q(身長): 170cm以上である
  - S(職業)：学生である
- クラス不変条件は3つの不変条件の論理積： Ｐ ∧ Ｑ ∧ Ｓ：
  - **Ｐ ∧ Ｑ ∧ Ｓ：身長170cm以上の日本人の学生**



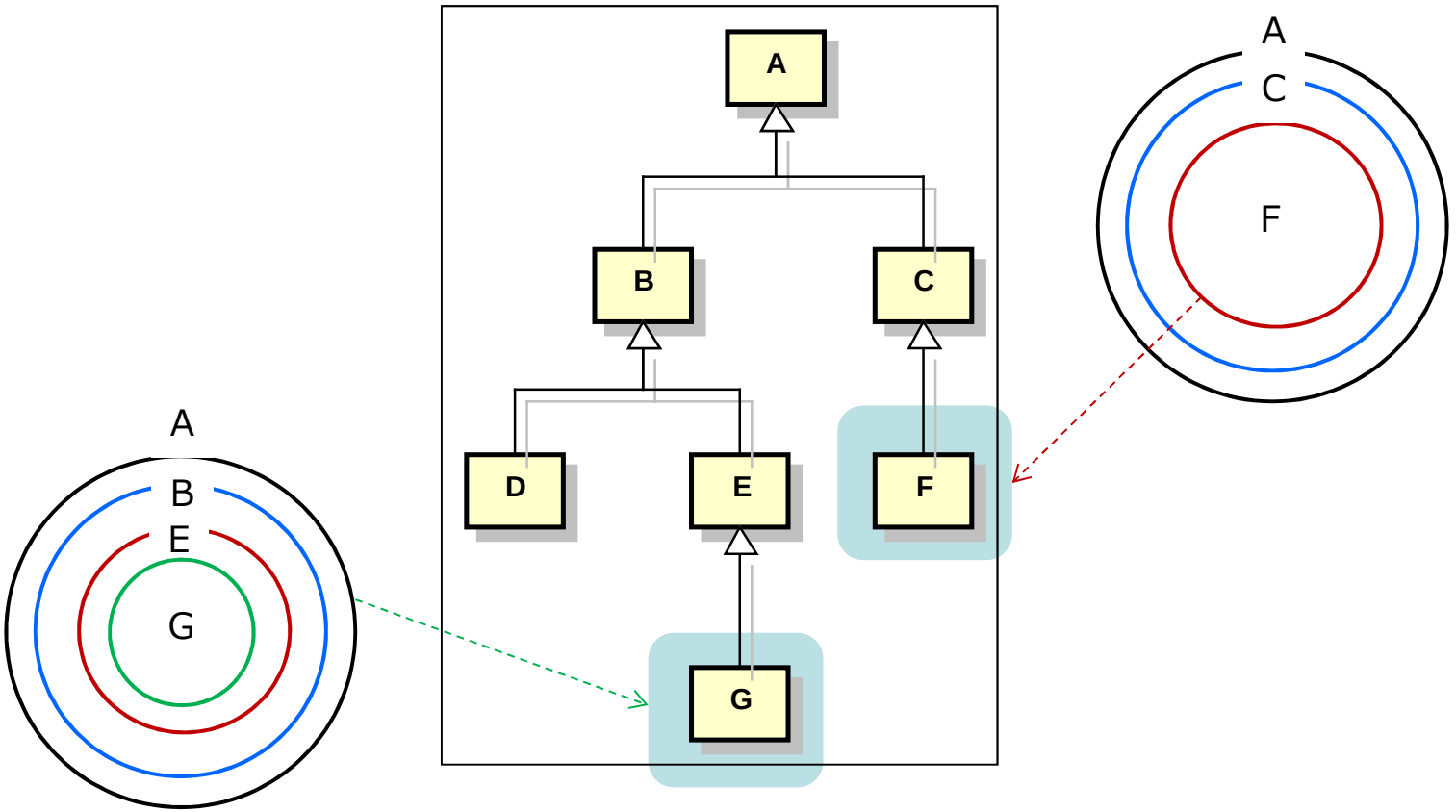


継承階層とクラス不変条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

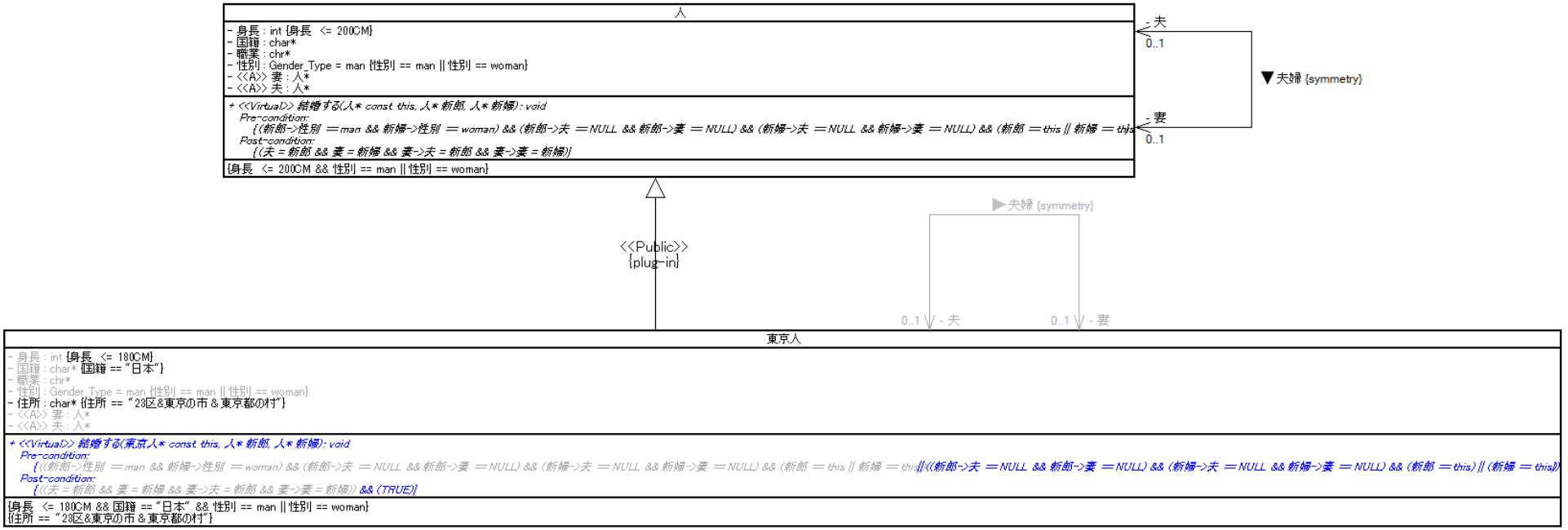
$SC_{invariant} \Rightarrow C_{invarinat}$

- スーパータイプの**不変条件(invariant)**はサブタイプにおいて**＜強める＞**ことはできるが**＜弱める＞**ことができない



サブタイプのクラス不変条件の例

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



(＊)HSCI Modeling tool
特許出願済み／Patent Pending

ローカル事前条件と実効事前条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

(操作の)事前条件と論理和

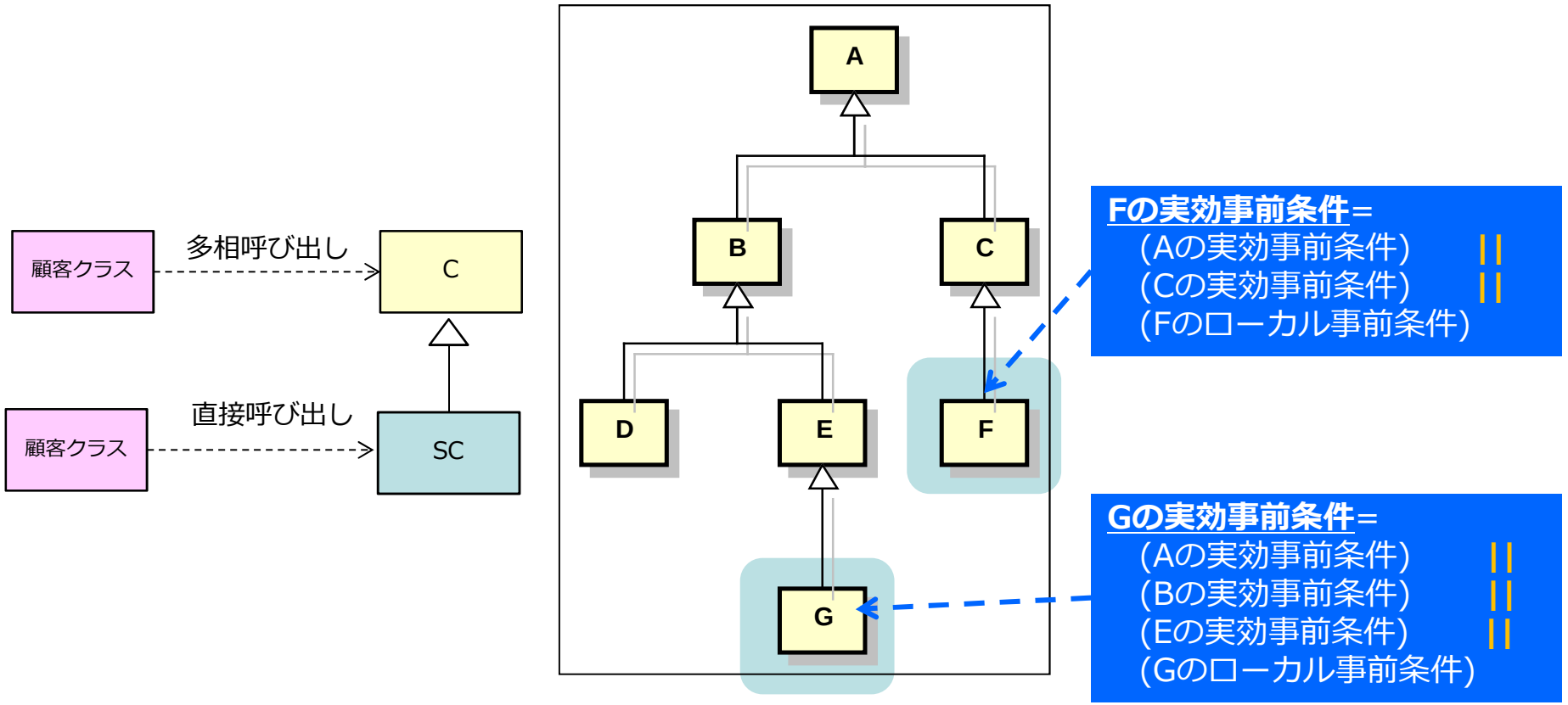
- サブタイプがスーパータイプの操作を**再定義(オーバーライド)**するときは,事前条件を**<弱くすること>**はできる
- 事前条件が**<弱い>**とは**条件の数**のこと
- サブタイプで再定義(オーバーライド)したとき：
  - **サブタイプの実効事前条件 = スーパータイプの実効事前条件 ||(論理和) サブタイプのローカル事前条件**
- サブタイプが再定義(オーバーライド)しないとき
  - **サブタイプの実効事前条件 = スーパータイプの実効事前条件**

継承階層と実効事前条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

操作事前条件：$C_{pre} \Rightarrow SC_{pre}$([plug-in]より弱い適合条件のとき)

- スーパータイプの**事前条件(pre-condition)**はサブタイプにおいて**＜弱める＞**ことはできるが**＜強める＞**ことはできない



ローカル事後条件と実効事後条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

(操作の)事後条件と論理積

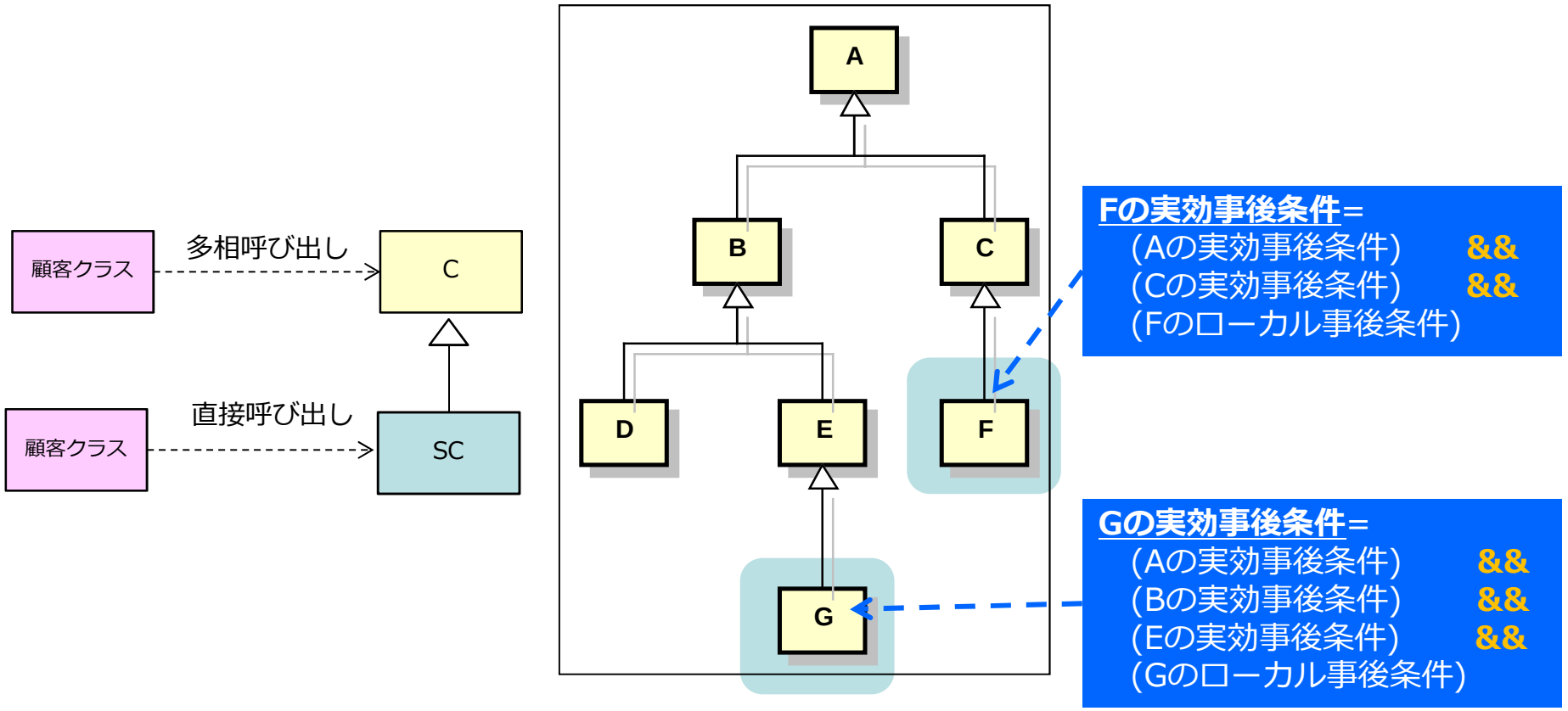
- サブタイプがスーパータイプの操作を**再定義(オーバーライド)**するときは,事後条件を**<強くすること>**はできる
- 事前条件が**<強い>**とは**条件の数**のこと
  - **サブタイプの実効事後条件 = スーパータイプの実効事後条件 &&(論理積) サブタイプのロカール事後条件**
- サブタイプが再定義(オーバーライド)しないときは,事後条件はそのまま継承される
  - **サブタイプの実効事後条件 = スーパータイプの実効事後条件**

継承階層と実効事後条件

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SCpost⇒Cpost([plug-in]より弱い適合条件のとき)

- スーパータイプの**事後条件(post-condition)**はサブタイプにおいて**＜強める＞**ことはできるが**＜弱める＞**ことができない



契約(表明)を実装する

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

Design By Contract™(契約による設計)

- C++やJavaではassertionによる実装
  - **クラス不変条件**
  - **属性の不変条件**
  - **操作の事前条件**
  - **操作の事後条件**
- **Plug-in**を満たす

厳格	Specification Match名称	Specification Match論理式
1	exact pre/post[Zaremski97][Chen00]	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
2	exact pre	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
2	exact post	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
4	plug-in[Zaremski97]	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
5	weak-plug-in([Penix99]) *[Zaremski97]ではguarded plug-inと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{pre} \wedge SC_{post} \Rightarrow C_{post})$
6	satisfies[Penix99] *[Chen00]ではrelaxed plug-in **[Fischer97]ではplug-in compatibilityと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (C_{pre} \wedge SC_{post} \Rightarrow C_{post})$

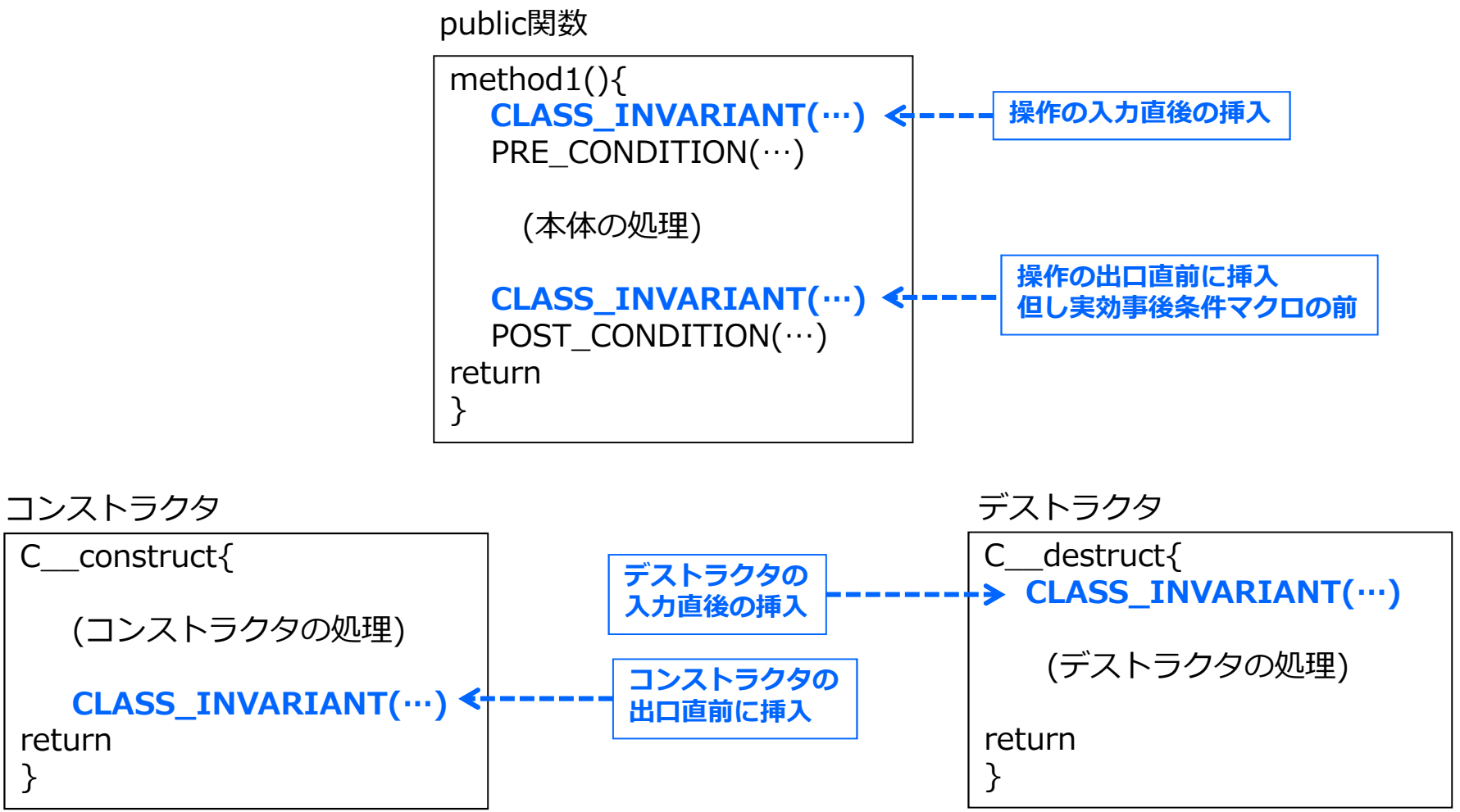
契約(表明)のコード定義位置

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

	操作種類	「クラス不変条件」	「実効事前条件」	「実効事後条件」
操作 入口	public操作	○	○	×
	コンストラクタ	×	×	×
	private操作 protected操作	×	△	×
	デストラクタ	○	×	×
	静的操作(クラス操作)	×	○	×
操作 出口	public操作	○	×	○
	コンストラクタ	○	×	×
	private操作 protected操作	×	×	△
	静的操作(クラス操作)	×	×	○

assertionマクロコードの挿入位置と順序

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.



ASSERT.hの例

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

assertionが[true]となる事をassertionを満たすと言う

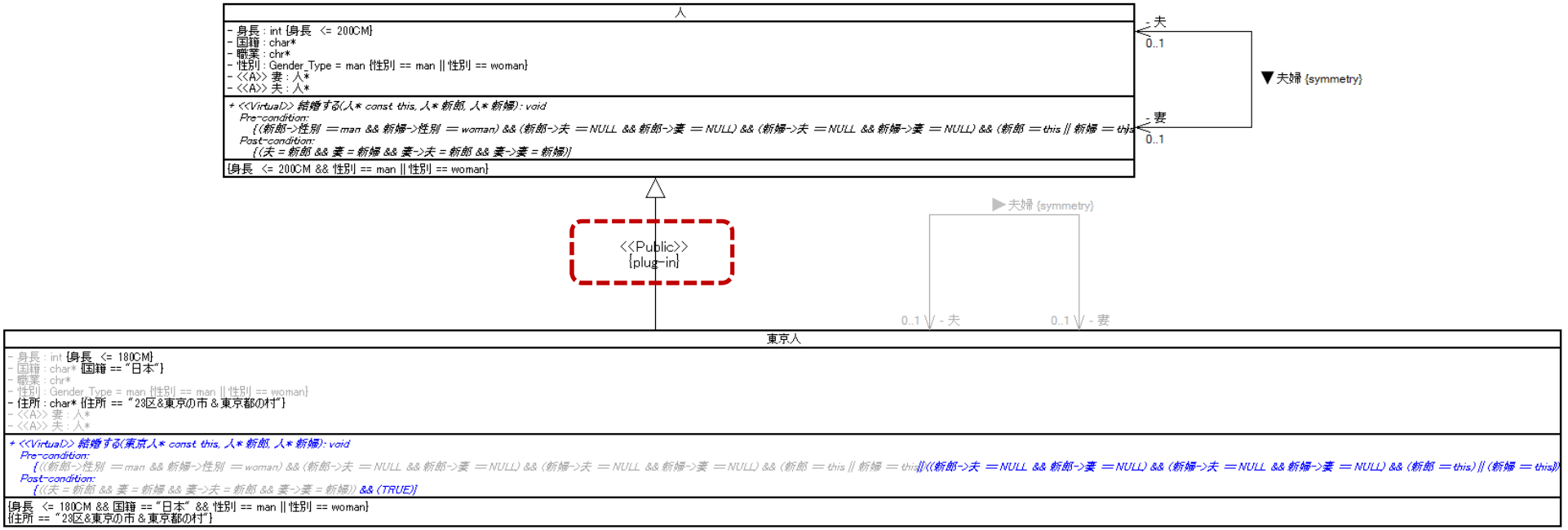
```
/*--- ASSERT.h---*/
#include <assert.h>

#define TRUE     0
#define FALSE    1
#define NDEBUG 0

#ifdef NDEBUG

    #define CLASS_INVARIAN(messaqe,condition)   assert((messaqe,(condition)))
    #define PRE_CONDITION(messaqe,condition)    assert((messaqe,(condition)))
    #define POST_CONDITION(messaqe,condition)  assert((messaqe,(condition)))

#else

    #define CLASS_INVARIAN(messaqe,condition)    /* Class Invariant isn't checked */
    #define PRE_CONDITION(messaqe,condition)     /* Pre-Condition isn't checked   */
    #define POST_CONDITION(messaqe,condition)   /* Post-Condition isn't checked */

#endif
```

- **クラス不変条件／事前条件／事後条件の実装方法は色々考案されている**
- **クラス不変条件／事前条件／事後条件をprotected関数で実装する方法もある**
- **今回は最も単純な方法を紹介する**

[Plug-in]の実装①

クラス不変条件:
SCinvariant⇒Cinvarinat
事前条件:
Cpre⇒SCpre
事後条件:
SCpost⇒Cpost

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.



(＊)HSCI Modeling tool
特許出願済み／Patent Pending

[Plug-in]の実装～スーパークラス「人」

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

```
/*****************************************************************************************************
  *[Visibility]: Public
  *[Type]: Virtual
  *[Description]:
  *[Pre-Conditions]: (新郎->性別 == man && 新婦->性別 == woman) && (新郎->夫 == NULL && 新郎->妻 == NULL) &&
                     (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this || 新婦 == this)
  *[Post-Conditions]: (夫 = 新郎 && 妻 = 新婦 && 妻->夫 = 新郎 && 妻->妻 = 新婦)
  *****************************************************************************************************/
void 人__結婚する(人* 新郎, 人* 新婦)
{
   /**--------------------クラス不変条件----------------------*/
   CLASS_INVARIANT("Violation of Class Invariant: 人.結婚する", (身長   <= 200CM && 性別 == man || 性別 == woman))

   /**----------------------事前条件-------------------------*/
   PRE_CONDITION("Violation of Pre Condition: 人.結婚する", ((新郎->性別 == man && 新婦->性別 == woman) && (新郎->夫 == NULL &&
                     新郎->妻 == NULL) && (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this || 新婦 == this)))

   夫 = 新郎;
   妻 = 新婦;

  /**--------------------クラス不変条件----------------------*/
  CLASS_INVARIANT("Violation of Class Invariant: 人.結婚する", (身長   <= 200CM && 性別 == man || 性別 == woman))

  /**----------------------事後条件-------------------------*/
  POST_CONDITION("Violation of Post Condition: 人.結婚する", ((夫 = 新郎 && 妻 = 新婦 && 妻->夫 = 新郎 && 妻->妻 = 新婦)))
}
```

(＊)前スライドのクラス図かコードを自動生成

[Plug-in]の実装～サブクラス「東京人」

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

```
/*****************************************************************************************************
  *[Visibility]: Public
  *[Type]: Virtual
  *[Description]:
  *[Pre-Conditions]: ((新郎->性別 == man && 新婦->性別 == woman) && (新郎->夫 == NULL && 新郎->妻 == NULL) &&
                      (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this || 新婦 == this)) || ((新郎->夫 == NULL && 新郎->妻 ==
                       NULL) && (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this) || (新婦 == this))
  *[Post-Conditions]:((夫 = 新郎 && 妻 = 新婦 && 妻->夫 = 新郎 && 妻->妻 = 新婦)) && (TRUE)
  *****************************************************************************************************/
void 東京人__結婚する(人* 新郎, 人* 新婦)
{
    /**--------------------クラス不変条件----------------------*/
    CLASS_INVARIANT("Violation of Class Invariant: 東京人.結婚する", (身長   <= 180CM && 国籍 == "日本" && 性別 == man || 性別 == woman
                        && 住所 == "23区&東京の市＆東京都の村"))

    /**----------------------事前条件-------------------------*/
    PRE_CONDITION("Violation of Pre Condition: 東京人.結婚する", (((新郎->性別 == man && 新婦->性別 == woman) && (新郎->夫 == NULL &&
                    新郎->妻 == NULL) && (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this || 新婦 == this)) || ((新郎->夫 ==
                    NULL && 新郎->妻 == NULL) && (新婦->夫 == NULL && 新婦->妻 == NULL) && (新郎 == this) || (新婦 == this))))

    夫 = 新郎;
    妻 = 新婦;

    /**--------------------クラス不変条件----------------------*/
    CLASS_INVARIANT("Violation of Class Invariant: 東京人.結婚する", (身長   <= 180CM && 国籍 == "日本" && 性別 == man || 性別 == woman
                        && 住所 == "23区&東京の市＆東京都の村"))

    /**----------------------事後条件-------------------------*/
    POST_CONDITION("Violation of Post Condition: 東京人.結婚する", (((夫 = 新郎 && 妻 = 新婦 && 妻->夫 = 新郎 && 妻->妻 = 新婦)) &&
                        (TRUE)))
}
```

(＊)前スライドのクラス図かコードを自動生成

assertionのメリット

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

- assertionはクラスの「契約」をクラス図やコードの中に出来るので,**テスト工数の削減**,**再利用を促進**し,**保守コストの削減**を助長する
  - 設計書の文書と異なり,「契約」の assertionは常に最新のコードおよび改定されたコードとの整合性を維持出来る
  - 「契約」にassertionはクラスや操作の本質的な仕様・責務を表現し,適切な箇所に記述できる
- assertionはオブジェクト指向の**テストに極めて有効**である:
  - カプセル化された状態と内部に隠ぺいされている変数を直接確認する事も出来る
  - assertionは,そうしないと観察の障害として伝播されない障害を検出する事が出来る
  - クラスの属性,クラス全体および操作の引数と返り値の定義域を明確にするのでテストケースの正確な情報源となる
- assertionの切り替えで**細かな制御が可能**となる：
  - クラス不変条件のみチェック(ON)する
  - 事前条件のみチェック(ON)する
  - 事後条件のみチェック(ON)する
  - 上記好きな組み合わせでassertionを有効(ON)できる
  - 特定のクラスや操作のみassertionを有効(ON)にできる
  - 全てのassertionをOFFにする

assertionの注意点

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

assertionの注意点

- assertionにアプリケーションエラーや例外を制御処理を含めない
- assertionにライブラリ操作やシステム･サービスからの戻り値をチェックしない
  - ただし,副作用がない操作はassertionで利用できる
- assertionはプライベート操作(private)でチェックしない
- 操作が例外をスローして異常終了したときに,事後条件とクラス不変条件のassertionチェックをしない
- 操作内に複数のreturn文がある場合は,各returnで,事後条件とクラス不変条件のassertionをチェックする
- テスト終了後はassertionを**OFF**にする
  - assertionコードは無効となりコードサイズ,処理速度も小さくなる
  - 簡単に開発モード(テストモード)・出荷モードの切り替えができる

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

Chapter5
《発展編》
タイプ(型)&モジュラー推論と安全な再利用

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

共変化(co-variance) vs. 無変化(no-variance) vs.逆変化(no-variance)

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

共変化(co-variance) vs.無変化(no-variance) vs.逆変化(no-variance)

- **共変化(co-variance)**
  - スーパークラスの引数の型をサブクラスの型で再定義できる
  - **柔軟な継承の利用**と**型安全が保証(タイプキャストが不要)**される
  - サブクラスでスーパークラスの引数の型を共変化させた場合は**LSPを満たさなくなる**(振る舞いサブタイプでは無くなる)
  - Eiffel,O2などが共変化も可能としている
- **無変化(no-variance)**
  - スーパークラスの引数の型をサブクラスで再定義できない
  - シグニチャは,スーパークラスから継承されるシグニチャと正確に一致しないといけない
  - 引数の型をサブクラスにすると操作はオーバーロードされる
  - **事前条件**と**動的タイプチェック**を利用する
  - C++／Javaなど多くの言語が採用
- **逆変化(no-variance)**
  - サブクラスで再定義された引数の型はスーパークラスの型にできる

共変返値と共変引数

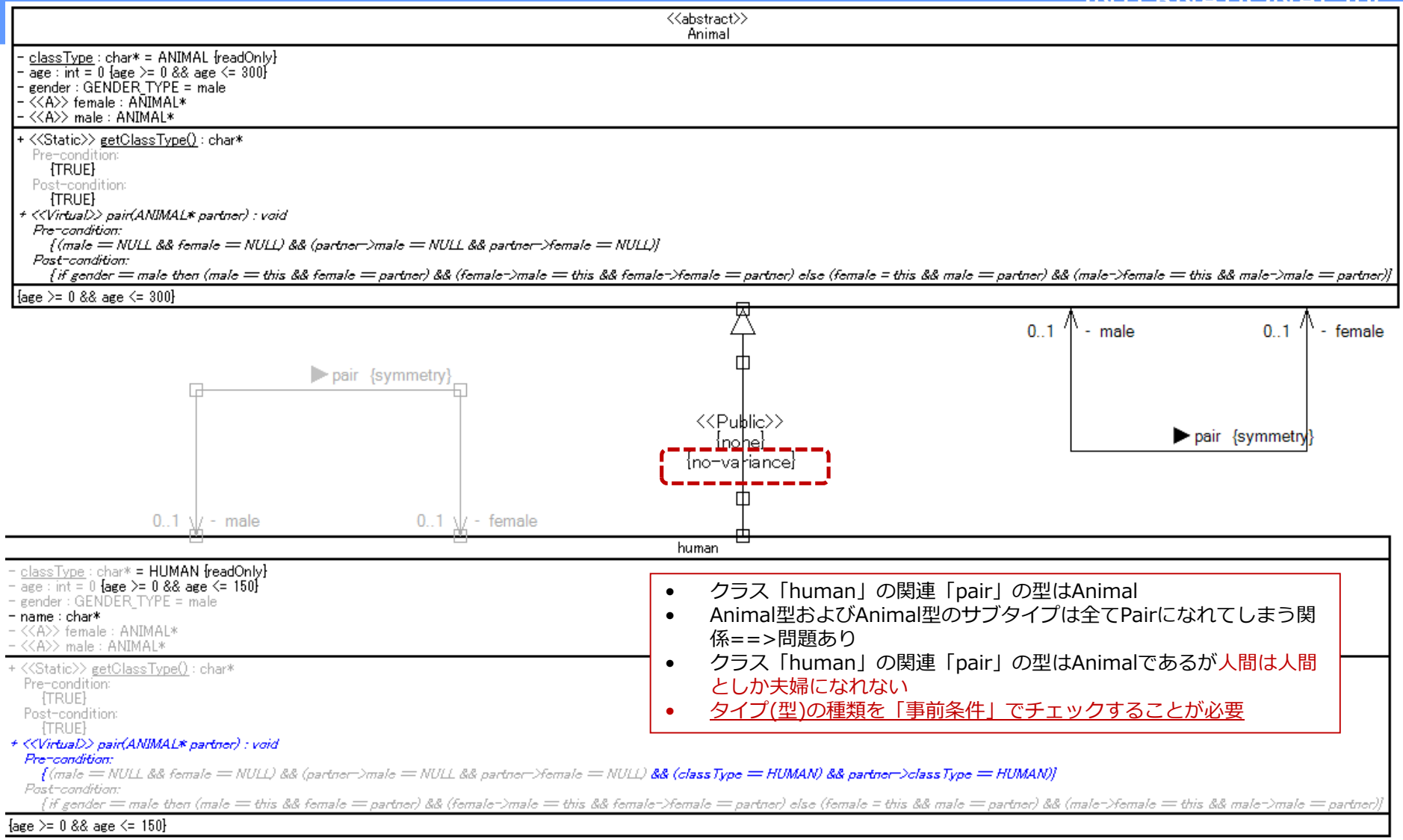
HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

共変化(co-variance)とタイプキャスト(type cast)

- シグニチャのタイプ(型)変化は,プログラム言語における**タイプ(型)の安全性の問題**である
- シグニチャのタイプ(型)変化は**タイプ(型)キャストと関係**し,シグニチャのタイプ(型)変化を使用出来なければ,タイプ(型)キャストを使用する必要が出てくる
- **共変返値：**
  - 操作をサブクラスでオーバーライドするときに,操作の返値の型をサブクラス型に置き換える事
- **共変引数：**
  - 操作をサブクラスでオーバーライドするときに,操作の引数の型をサブクラス型に置き換える事
  - 特定の型についての振舞いのみオーバーライドさせる

無変化(no-variance)のモデル

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc

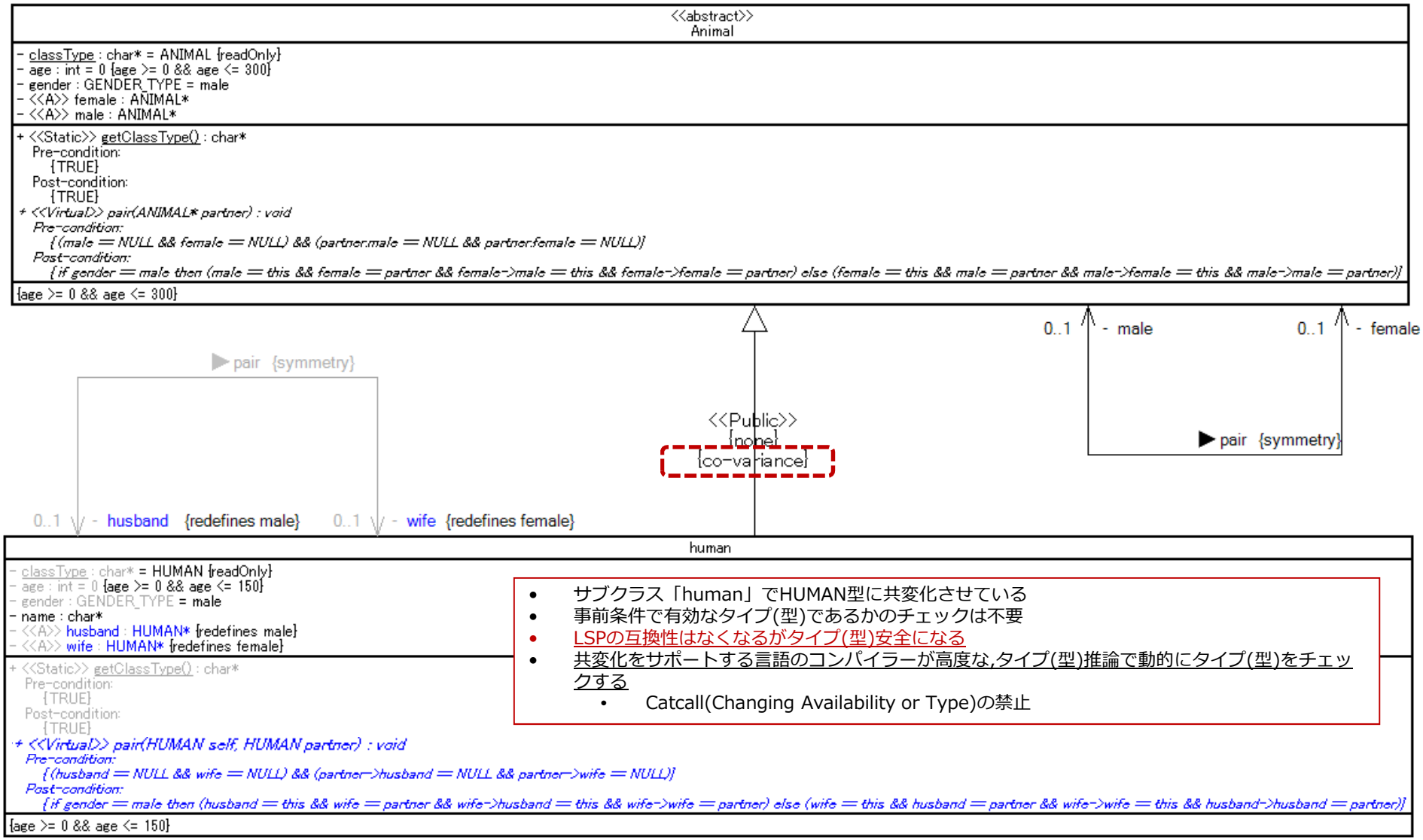


- クラス「human」の関連「pair」の型はAnimal
- Animal型およびAnimal型のサブタイプは全てPairになれてしまう関係==>問題あり
- クラス「human」の関連「pair」の型はAnimalであるが人間は人間としか夫婦になれない
- タイプ(型)の種類を「事前条件」でチェックすることが必要

(＊)HSCI Modeling tool
特許出願済み／Patent Pending

共変化(co-variance)モデル

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc



- サブクラス「human」でHUMAN型に共変化させている
- 事前条件で有効なタイプ(型)であるかのチェックは不要
- LSPの互換性はなくなるがタイプ(型)安全になる
- 共変化をサポートする言語のコンパイラーが高度な,タイプ(型)推論で動的にタイプ(型)をチェックする
  - Catcall(Changing Availability or Type)の禁止

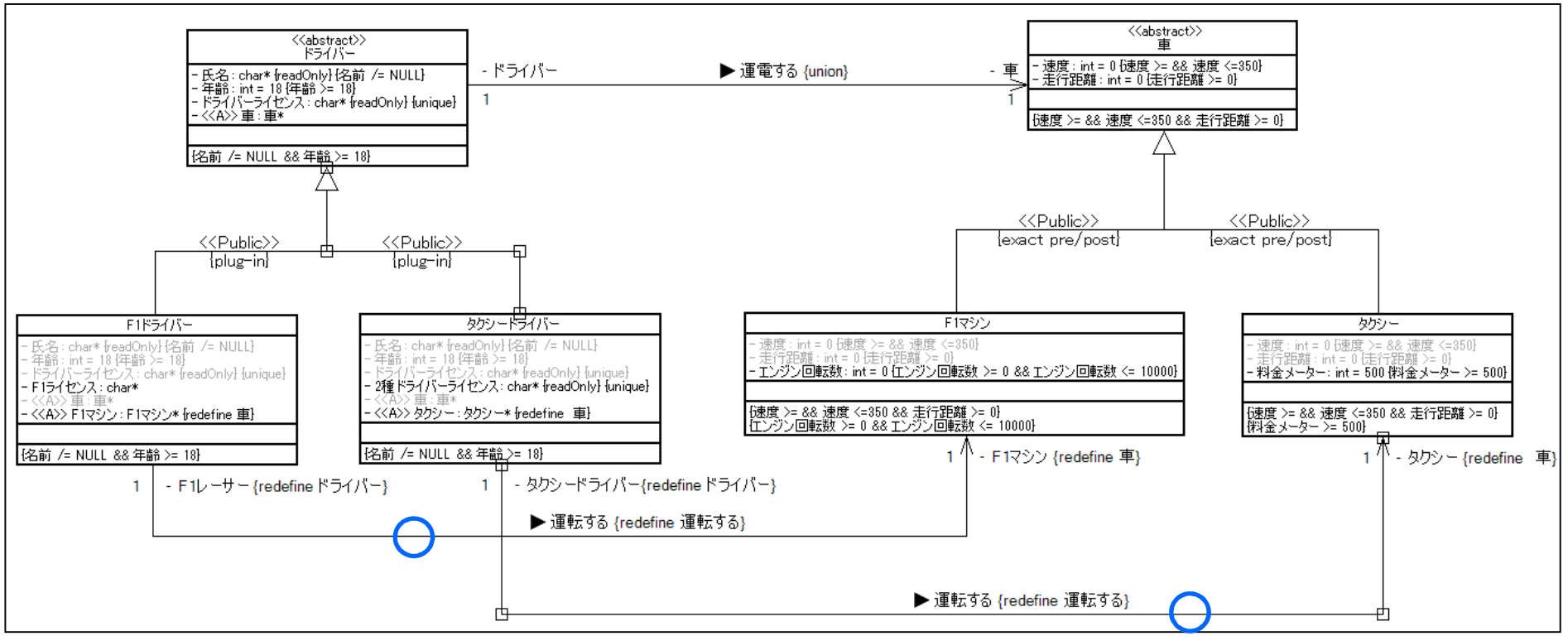
(＊)HSCI Modeling tool
特許出願済み／Patent Pending

共変化(co-variance)とデザインパターン

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

デザインパターンの多くは共変化モデル

- リスコフ置換原則(LSP)に完全準拠する事は基本だが,時として常に準拠が必要とは限らない
- **共変化モデル(LSP非準拠)**の方が実開発で多く利用される研究報告も多い
  - 継承関係で**共変的なモデル**は,タイプ(型)安全のために必ず事前条件でのチェックが必要
- Observerパターン／AbstractFactoryパターン／Bridgeパターン／etc,は**共変的モデル**



開放閉鎖原則とModular Reasoning

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

Modular Reasoning(モジュラー推論)
• **振る舞いサブタイプの継承**で満たすべき重要な要件は**「モジュラー推論」を維持**すること • モジュラー推論により,継承階層やクラスライブラリのクラスのサブクラス／コンポーネント**は拡張に開いていることを保証**する • モジュラー推論により,継承階層やクラスライブラリに新たなクラスのサブクラス／コンポーネントも,変更されていない**クライアントコードに対しては再検証する必要が無いことを保証**する

Modular Reasoningによる再利用判定

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

Modular Reasoning(モジュラー推論)で再利用を保証する
• 新しいサブクラスを追加したとき**意味的な整合性**のチェックが必要となる • 予期しない振る舞いを**モジュラー推論で防止する** • スーパークラスのオブジェクトを使用するクライアントコードが,継承階層内のサブクラスを利用しても**意味的整合性が保持**することを保証する • サブクラスによってオーバーライドされた操作群が,スーパークラスの仕様を満足するなら,**サブクラスは再利用対象として「仕様整合性」を満たす**事を意味する

Specification Matchによるモジュラー推論

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

厳しさ	Specification Match名称	Specification Match論理式
1	exact pre/post[Zaremski97][Chen00]	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
2	exact pre	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
2	exact post	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
4	plug-in[Zaremski97]	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
5	weak-plug-in([Penix99]) *[Zaremski97]ではguarded plug-inと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{pre} \wedge SC_{post} \Rightarrow C_{post})$
6	satisfies[Penix99] *[Chen00]ではrelaxed plug-in **[Fischer97]ではplug-in compatibilityと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (C_{pre} \wedge SC_{post} \Rightarrow C_{post})$

クラスやコンポーネントの再利用に対して常に保守的な置換推論

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

厳しさ	Specification Match名称	Specification Match論理式
1	exact pre/post[Zaremski97][Chen00]	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
2	exact pre	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
2	exact post	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
4	plug-in[Zaremski97]	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
5	weak-plug-in([Penix99]) *[Zaremski97]ではguarded plug-inと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{pre} \wedge SC_{post} \Rightarrow C_{post})$
6	satisfies[Penix99] *[Chen00]ではrelaxed plug-in **[Fischer97]ではplug-in compatibilityと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (C_{pre} \wedge SC_{post} \Rightarrow C_{post})$

保守的かつ追加機能拡張したクラスやコンポーネント置換が可能な推論

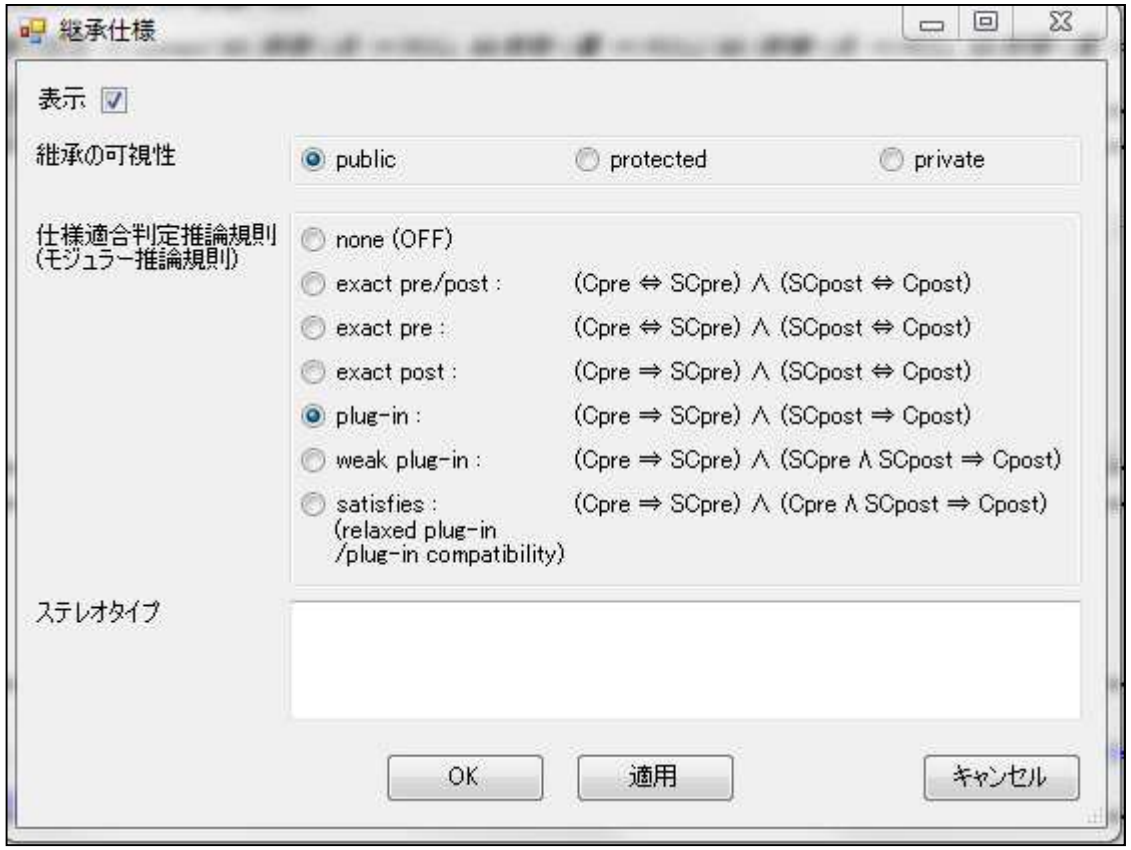
HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

厳しさ	Specification Match名称	Specification Match論理式
1	exact pre/post[Zaremski97][Chen00]	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
2	exact pre	$(C_{pre} \Leftrightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
2	exact post	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Leftrightarrow C_{post})$
4	plug-in[Zaremski97]	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{post} \Rightarrow C_{post})$
5	weak-plug-in([Penix99]) *[Zaremski97]ではguarded plug-inと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (SC_{pre} \wedge SC_{post} \Rightarrow C_{post})$
6	satisfies[Penix99] *[Chen00]ではrelaxed plug-in **[Fischer97]ではplug-in compatibilityと呼ばれる	$(C_{pre} \Rightarrow SC_{pre}) \wedge (C_{pre} \wedge SC_{post} \Rightarrow C_{post})$

設計目的に最適な条件式を簡単に設定・変更

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

- 目的や用途に合わせて自由に条件式を設定できる
- 条件式に準拠した「事前条件」「事後条件」を各操作に自動で適切な位置に生成・定義する
- 条件式の選択を変えれば、自動で「事前条件」「事後条件」を変更できる
- なお継承種類を《protected》《private》を選択したときは,実装継承になるため置換原則満たさないので,条件式の設定機能は自動でOFFになる



(＊)HSCI Modeling tool
特許出願済み／Patent Pending

条件式の表示・コード生成が自由な組み合わせで設定・変更可能

HASHIMOTO SOFTWARE CONSULTING INTERNATIONAL Inc.

- 「クラス不変条件」「事前条件」「事後条件」を各操作に自動で適切な位置に生成・定義する
- チェックを外しOFFにすれば，「クラス不変条件」「事前条件」「事後条件」の条件式のコードを生成しない

(＊)HSCI Modeling tool
特許出願済み／Patent Pending

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

HSCI Professional Learning Course Series

WRAP-UP
《いざない》
設計コンテストにGO!!

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

SEIPartner
CMMI

ETロボコンで先進設計にチャレンジ

HASHIMOTO
SOFTWARE
CONSULTING
INTERNATIONAL Inc.

ETロボコンでチャレンジ!!

- 普段できない設計アプローチでチャレンジ
- 品質と生産性を意識した開発設計アプローチでチャレンジ
- **ソフトウエア開発の新大陸**を求めて航海に出かけよう!!

ETロボコン 2014 Embedded Technology Software Design Robot Contest