Real v. Not

- Language notation RULES shaped by:
  - Reality (observed RULES implemented in notation)
  - Contrivance (artificial or invented “rules”)
- Example: Math notation RULES → Observed
- Observed RULES → Useful Math (v. Useless)
- Math RULES are not:
  - Invented → You could make this stuff up, but …
  - Artificial → To what purpose (useful semantic)?
  - Contrived → The results would be disastrous!

RULES: Good, Bad, Ugly

- **if** [language] = contrived **then** [result]
- Where:
  - Good = Based on observed rules of reality
  - Bad = Somewhat good
  - Ugly = Utterly invented or made up
- Results?
  - Good: Elegant, highly useful, LESS bugs!
  - Bad/Ugly: Difficult, questionably useful, BUGS!

All languages “Create”

- How?
  - By special “Creation Procedures”
    (In Java = “Constructors”)
  - Following special RULES
- What are they?
  - “Creation instructions” → makes object & sets fields
  - Language notation to ID “creation instructions”

PANCAKE Creation RULES

- Q: How does one make pancakes?
  - A: Recipe (written or remembered)
- Q: Is there only one recipe for pancakes?
  - A: NO! There are many!
- Q: If we make “Class PANCAKE” and the constructors are “recipes”, do we expect many creation procedures?
  - A: YES, WE DO!

THEN: Why does Java have only ONE?

Java Constructor “rules”

- Constructor name MUST = Class name (why?)
- Dif constructors = same name/dif args (why?)*
- “super” must be in first line (why?)*
- Many other irrational “rules”!
- Java is NOT the only one!

** NO logical or rational reason!*

Consequences: Java

- Multiple “constructors” / same name
  - Severe lack of semantics
  - Small semantics → Big Confusion → Big Bugs

What if the only thing you see is this code?

How can you tell the difference (semantically) between them?

Answer: You cannot! You MUST look at the implementation of each "form" of "constructor" to know because only the actual implementation code reveals the purpose of each form of "constructor".

```
public class Platypus {
        String name;
        Platypus(String input) {
                name = input;
        }
        Platypus() {
                this("John/Mary Doe");
        }
        public static void main(String args[]) {
                Platypus p1 = new Platypus("digger");
                Platypus p2 = new Platypus();
        }
}
```

Property or feature being affected

Constructors

Creation calls #1 (w/"String") and #2 (w/o args)

  - “Platypus” alone tells no story about creation!!!
  - Okay, I pass a String; so what? That means what?

PANCAKE CLASS

- NOTE:
  - One class: PANCAKE
  - Many recipes (creations)
  - Constrained access:
    - HOME_BREW
    - CHEFS
  - Clear!
  - Elegant!

```
class
    PANCAKE

create {HOME_BREW}
    make_deairas_method,
    make_brads_method,
    make_bradleys_method,
    make_michaels_method,
    make_larrys_method

create {CHEFS}
    make_alton_brown_method,
    make_gordon_ramsey
```

```
class
    HOME_BREW

feature -- Pancakes

    pancakes: ARRAYED_LIST [PANCAKE]
        do
            create Result.make (5)
            Result.force (create {PANCAKE}.make_deairas_method)
            Result.force (create {PANCAKE}.make_brads_method)
            Result.force (create {PANCAKE}.make_bradleys_method)
            Result.force (create {PANCAKE}.make_michaels_method)
            Result.force (create {PANCAKE}.make_larrys_method)
        end

end
```

Real v. Not

- Language notation RULES shaped by:
  - Reality (observed RULES implemented in notation)
  - Contrivance (artificial or invented “rules”)
- Example: Math notation RULES → Observed
- Observed RULES → Useful Math (v. Useless)
- Math RULES are not:
  - Invented → You could make this stuff up, but ...
  - Artificial → To what purpose (useful semantic)?
  - Contrived → The results would be disastrous!

RULES: Good, Bad, Ugly

- if [language] = contrived then [result]
- Where:
  - Good = Based on observed rules of reality
  - Bad = Somewhat good
  - Ugly = Utterly invented or made up
- Results?
  - Good: Elegant, highly useful, LESS bugs!
  - Bad/Ugly: Difficult, questionably useful, BUGS!

All languages “Create”

- How?
  - By special “Creation Procedures”
    (In Java = “Constructors”)
  - Following special RULES
- What are they?
  - “Creation instructions” → makes object & sets fields
  - Language notation to ID “creation instructions”

PANCAKE Creation RULES

- Q: How does one make pancakes?
  - A: Recipe (written or remembered)
- Q: Is there only one recipe for pancakes?
  - A: NO! There are many!
- Q: If we make “Class PANCAKE” and the constructors are “recipes”, do we expect many creation procedures?
  - A: YES, WE DO!

THEN: Why does Java have only ONE?

Java Constructor “rules”

- Constructor name MUST = Class name (why?)
- Dif constructors = same name/dif args (why?)*
- “super” must be in first line (why?)*
- Many other irrational “rules”!
- Java is NOT the only one!

** NO logical or rational reason!*

Consequences: Java

- Multiple “constructors” / same name
  - Severe lack of semantics
  - Small semantics → Big Confusion → Big Bugs

What if the only thing you see is this code?

How can you tell the difference (semantically) between them?

Answer: You cannot! You MUST look at the implementation of each "form" of "constructor" to know because only the actual implementation code reveals the purpose of each form of "constructor".

```
public class Platypus {
        String name;
        Platypus(String input) {
                name = input;
        }
        Platypus() {
                this("John/Mary Doe");
        }
        public static void main(String args[]) {
                Platypus p1 = new Platypus("digger");
                Platypus p2 = new Platypus();
        }
}
```

Property or feature being affected

Constructors

Creation calls #1 (w/"String") and #2 (w/o args)

  - “Platypus” alone tells no story about creation!!!
  - Okay, I pass a String; so what? That means what?

PANCAKE CLASS

- NOTE:
  - One class: PANCAKE
  - Many recipes (creations)
  - Constrained access:
    - HOME_BREW
    - CHEFS
  - Clear!
  - Elegant!

```
class
    PANCAKE

create {HOME_BREW}
    make_deairas_method,
    make_brads_method,
    make_bradleys_method,
    make_michaels_method,
    make_larrys_method

create {CHEFS}
    make_alton_brown_method,
    make_gordon_ramsey
```

```
class
    HOME_BREW

feature -- Pancakes

    pancakes: ARRAYED_LIST [PANCAKE]
        do
            create Result.make (5)
            Result.force (create {PANCAKE}.make_deairas_method)
            Result.force (create {PANCAKE}.make_brads_method)
            Result.force (create {PANCAKE}.make_bradleys_method)
            Result.force (create {PANCAKE}.make_michaels_method)
            Result.force (create {PANCAKE}.make_larrys_method)
        end

end
```