Code Generation via Interactive Source-to-Source Transformations

Marat Boshernitsan, Susan L. Graham
{maratb,graham}@cs.berkeley.edu

Computer Science Division, EECS
University of California, Berkeley

HARMONIA
http://harmonia.cs.berkeley.edu

GPCE – STS Workshop - 2004

Ad-hoc Code Generation

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {
   }
}
```

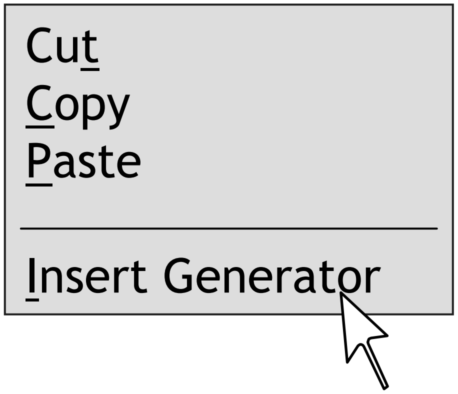
Before

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {
      string.print();
      value.print();
   }
}
```

After

Interactive Manipulation

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```



Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

Interactive Manipulation

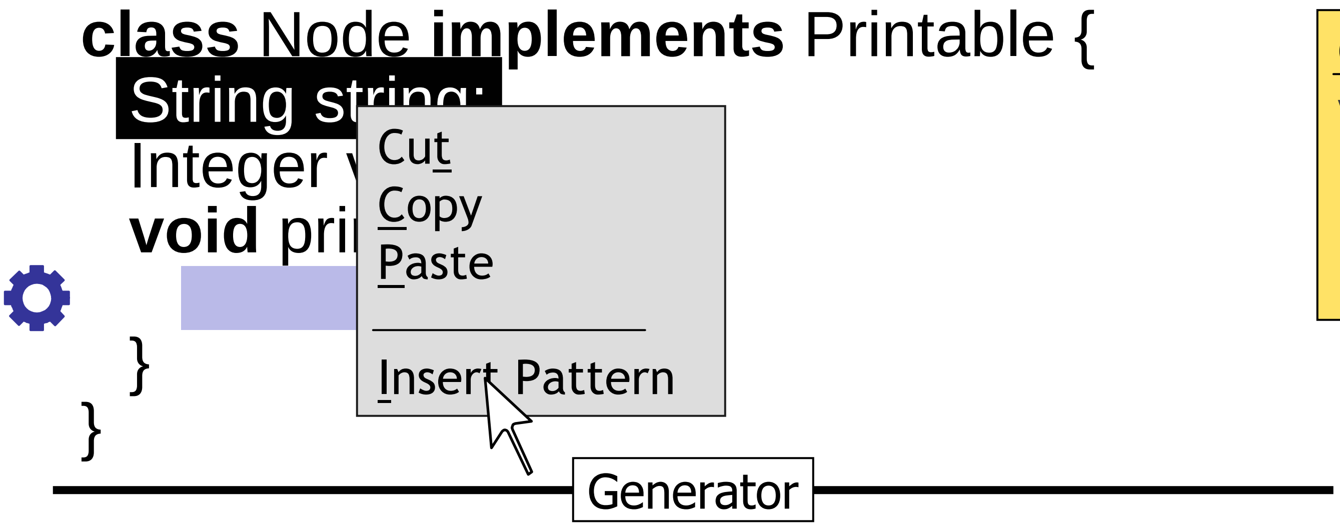
```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

Generator

Generalization from Example



```
class Node implements Printable {
  String string;
  Integer v
  void pri

  }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

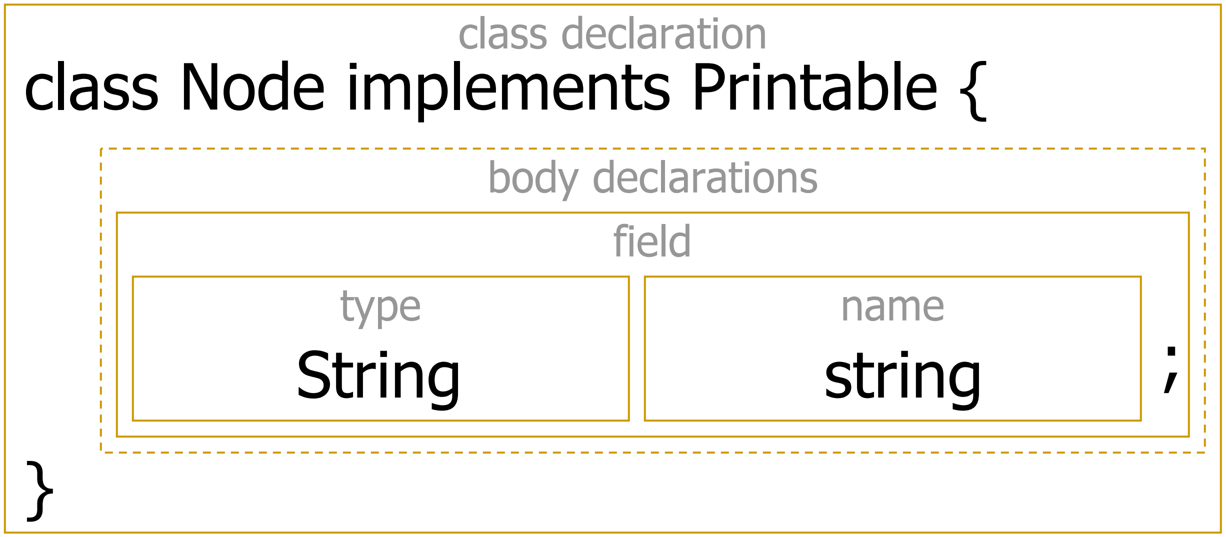
Hybrid Pattern Language

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

Generator



class declaration

class Node implements Printable {

body declarations

field

type

String

name

string

;

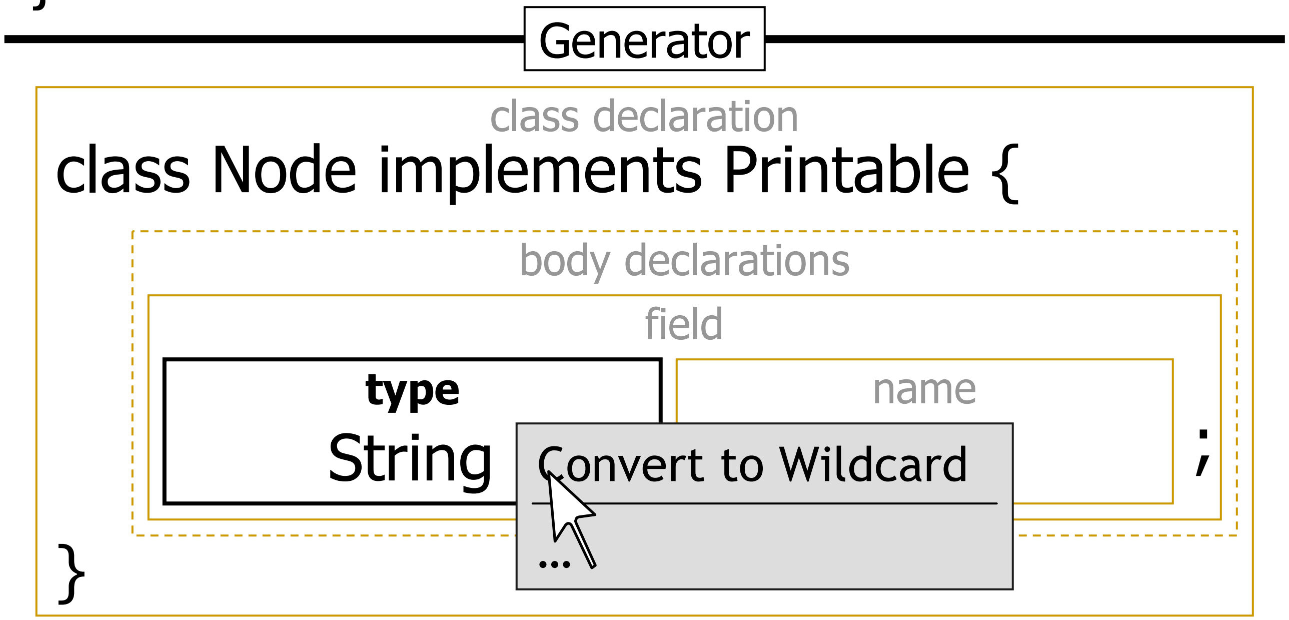
}

Conceptual Language Model

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```



Generator

class declaration

class Node implements Printable {

body declarations

field

type

String

name

;

Convert to Wildcard

...

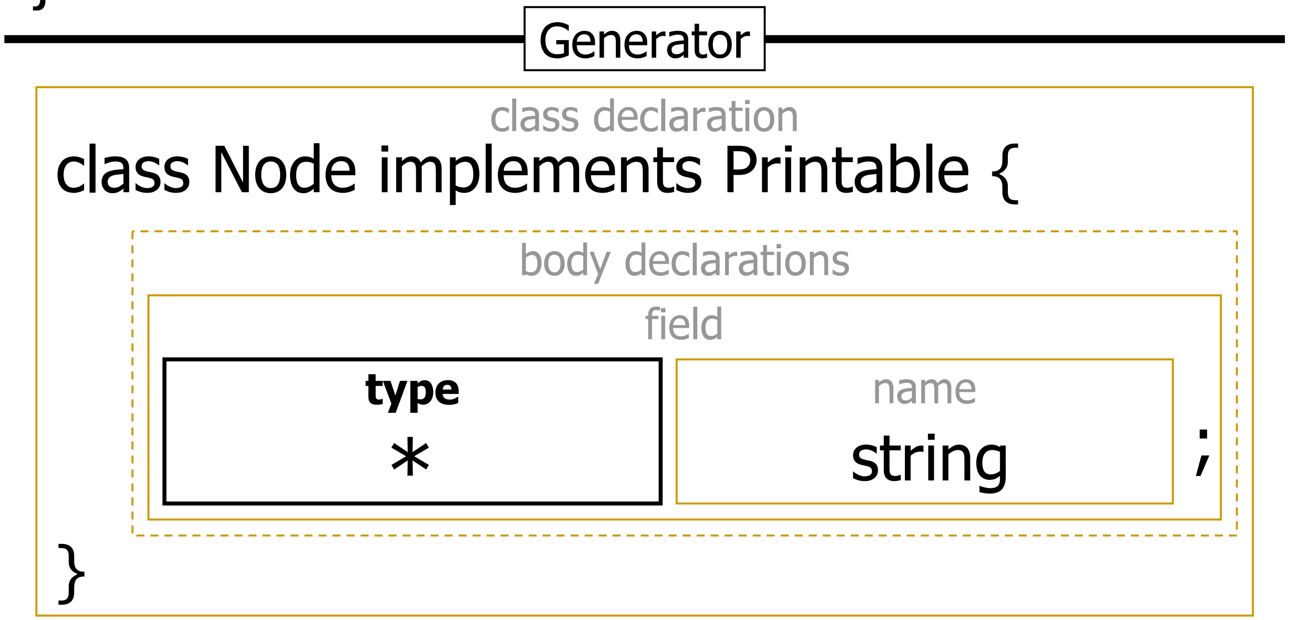
}

Conceptual Language Model

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```



Generator

class declaration

class Node implements Printable {

body declarations

field

type

*

name

string

;

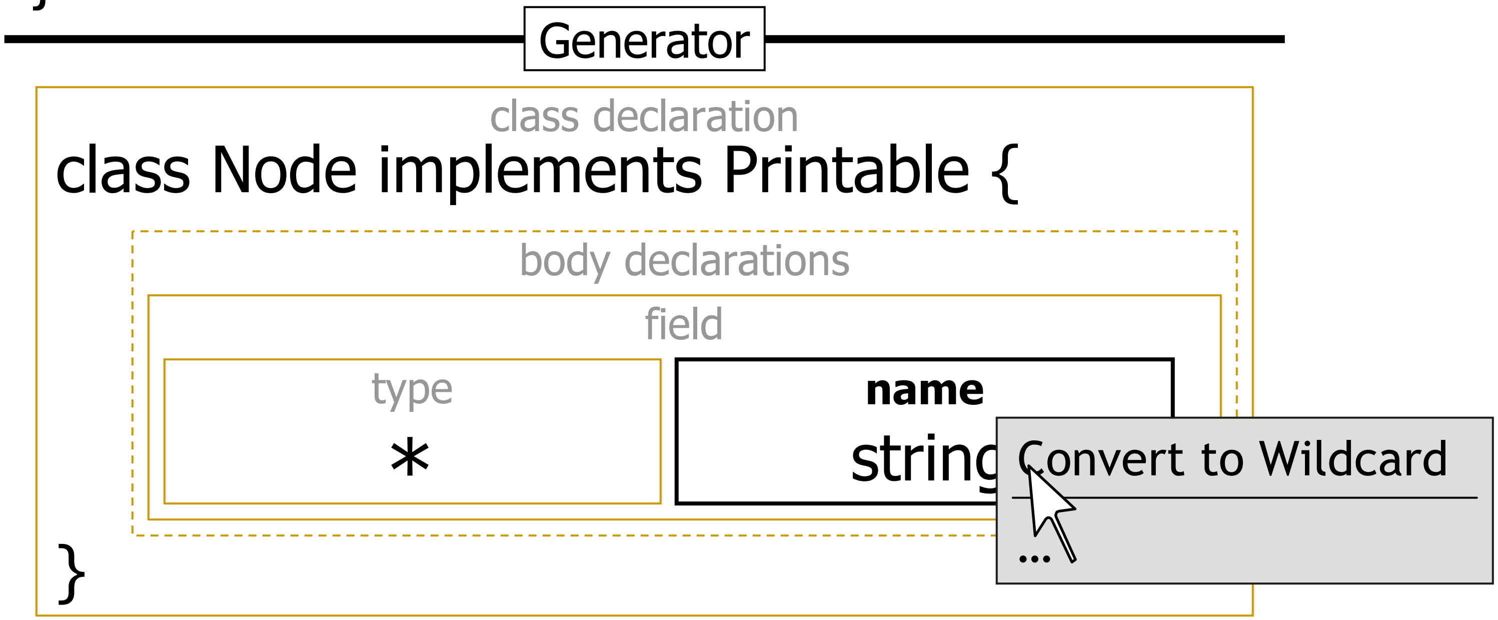
}

Incremental Refinement

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

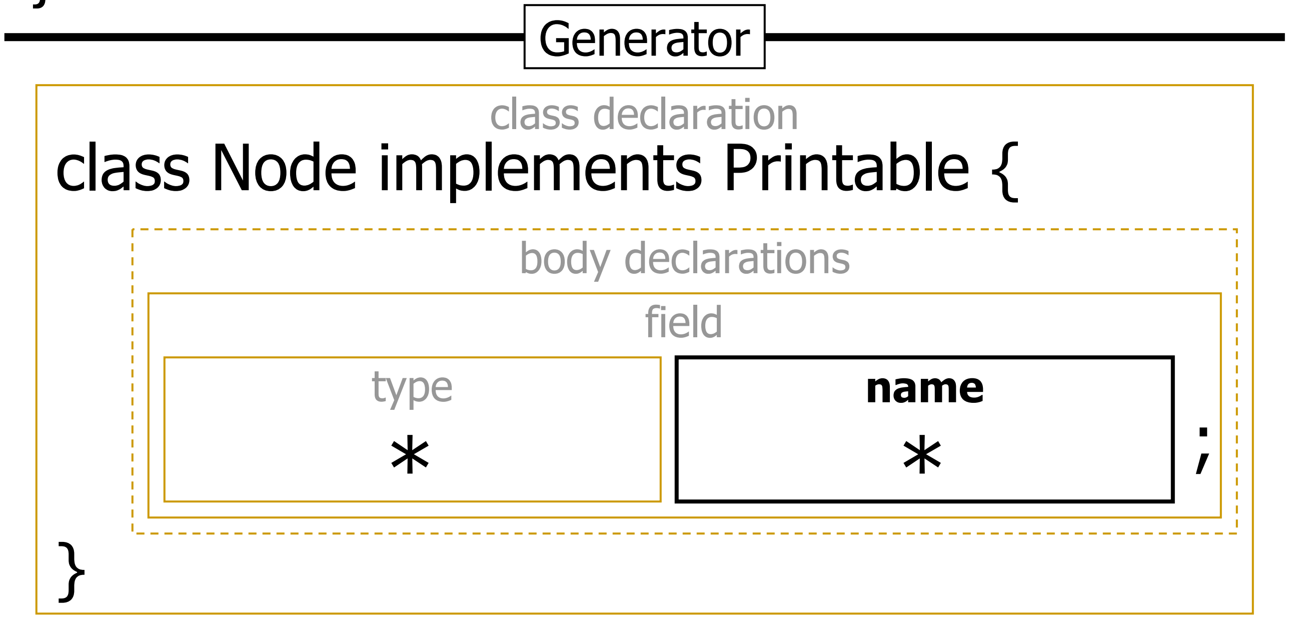


Immediate Feedback

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```



Generator

class declaration

class Node implements Printable {

body declarations

field

type

*

name

*

;

}

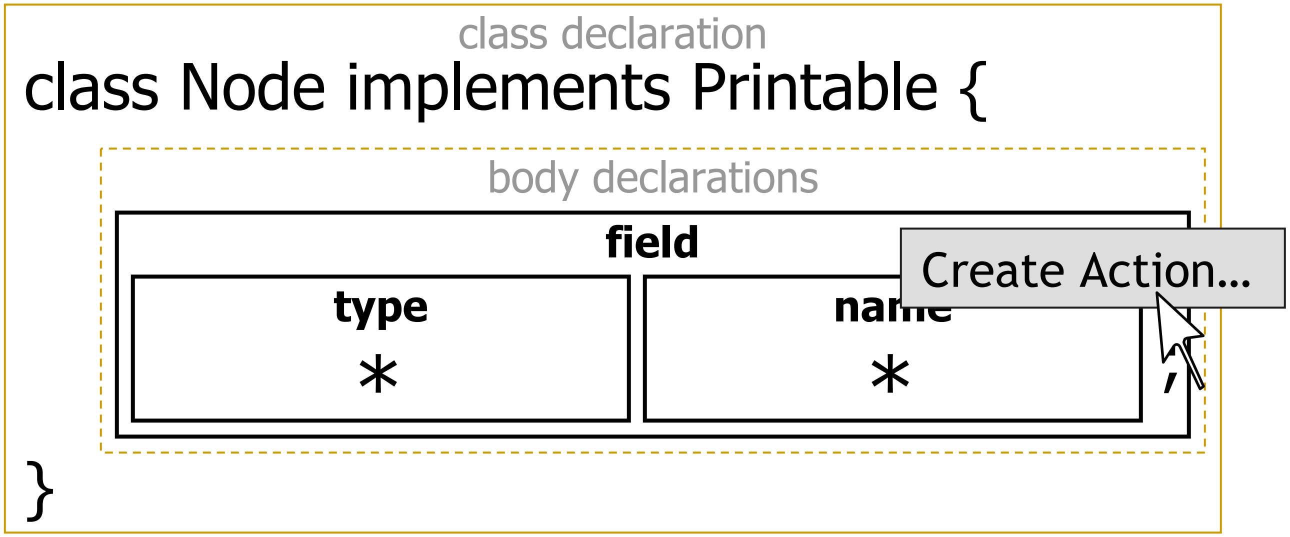
Direct Manipulation

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

Generator

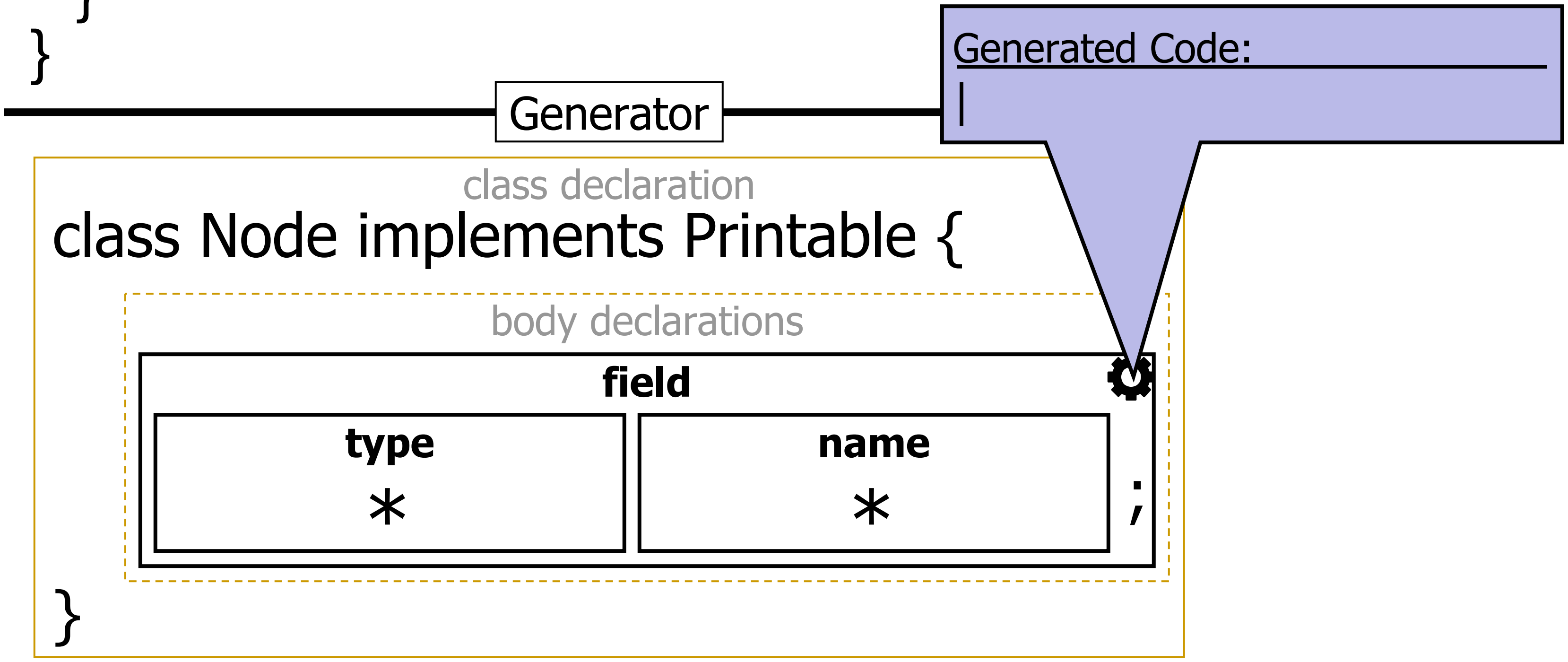


Direct Manipulation

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {

   }
}
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

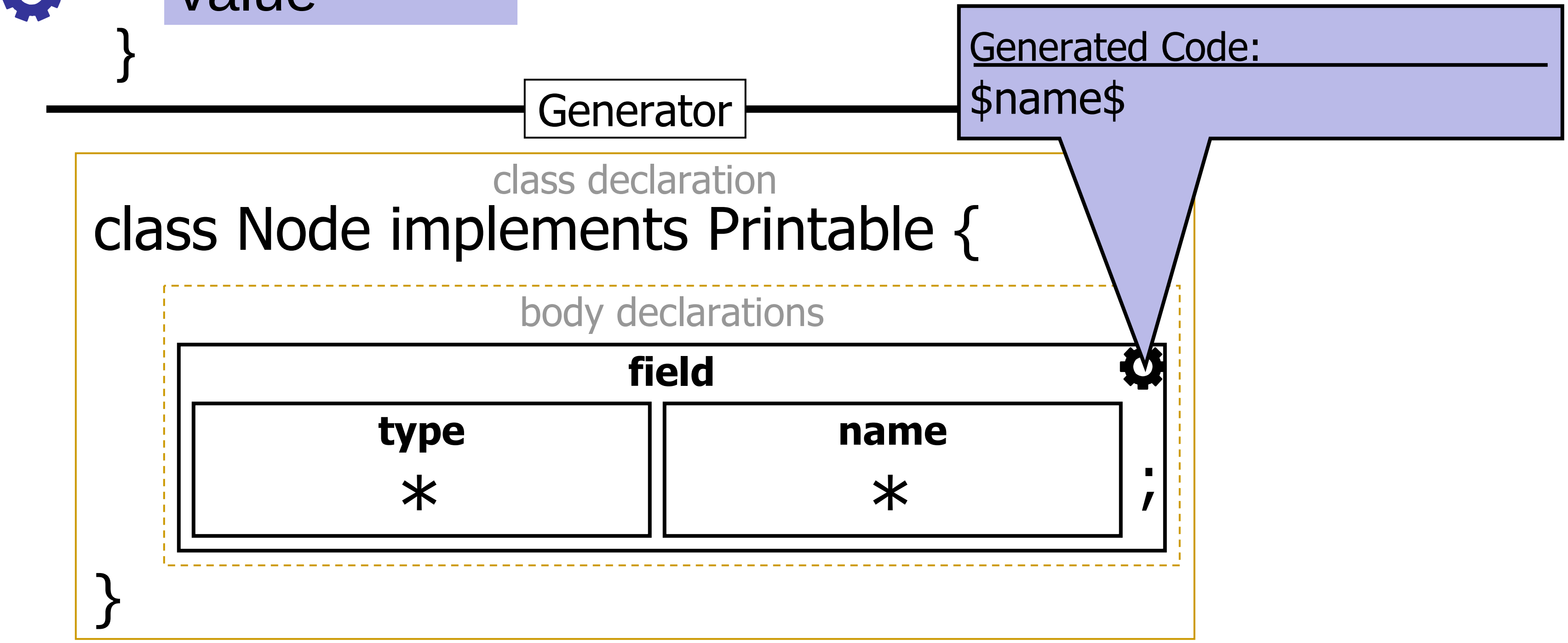


Incremental Visualization

```
class Node implements Printable {
   String string;
   Integer value;
   void print() {
      string
      value
   }
```

Our Goal:

```
void print() {
   string.print();
   value.print();
}
```



Generator

Generated Code:

$name$

class declaration

class Node implements Printable {

body declarations

field

type

*

name

*

;

}

Incremental Visualization

```
class Node implements Printable {
  String string;
  Integer value;
  void print() {
      string.print();
      value.print();
  }
```

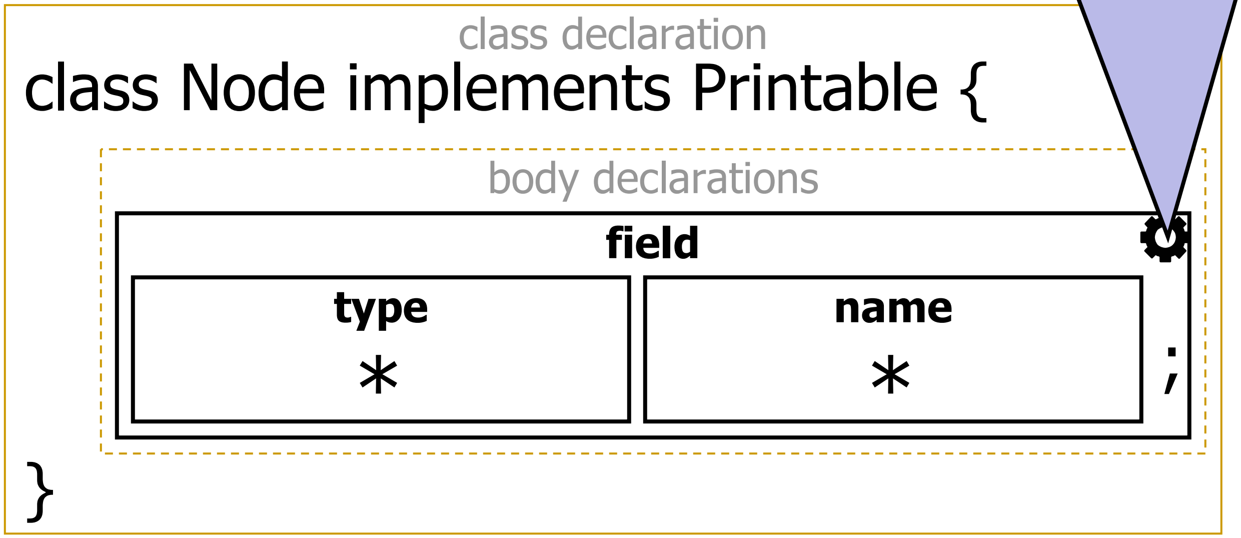
Our Goal:

```
void print() {
   string.print();
   value.print();
}
```

Generator

Generated Code:

```
$name$.print();
```



Lightweight Transformations

- Lightweight = “Ad-hoc”
- Hybrid textual/visual pattern language
  - End-programmer != tool builder
- Interactive Transformation Development
  - System scaffolds initial construction
  - Interface encourages experimentation
  - Immediate feedback makes execution of transformations transparent

Recurring theme: end-programmer usability!

Current Status

- Java pattern language is 80% done
  - Design inspired by experiments
- Interactive transformation environment
  - Plugs into Eclipse JDT
  - Utilizes the Harmonia framework
  - Can be an interface to traditional transformation tools

Evaluation

- Expressiveness: power to express common transformations
- Usability: can programmers use it?
  - Do they understand our vocabulary?
  - How intuitive is the pattern structure?
  - How comfortable is the process of developing transformations?
- Usability Metrics
  - Performance on sample tasks
  - Learning time
  - Kinds of mistakes

Conclusion

- Major Contributions
  - Makes ad-hoc transformations a standard editing paradigm for source manipulation
  - Prototypes a tool for lightweight source code transformations
  - Validates design methodology for building transformation languages and interfaces
- Poster at the OOPSLA poster session

maratb@cs.berkeley.edu
http://harmonia.cs.berkeley.edu