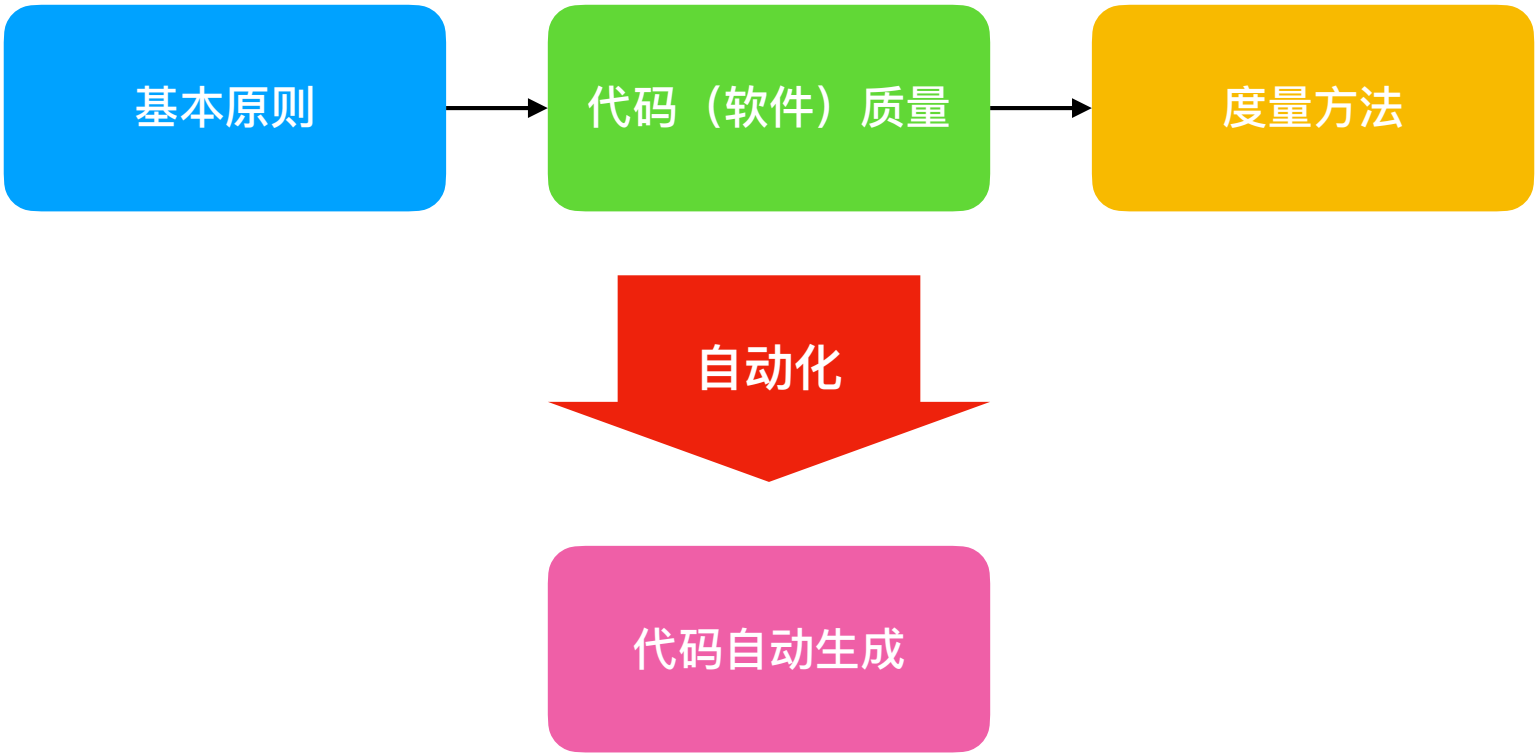
北京科技大学
University of Science and Technology Beijing

软件工程

编码与实现

编码与实现



编码的基本原则

基本原则

- 选择合适的编程语言
- 良好的编码风格和习惯

变量、函数、类型命名合理、有意义

充分进行注释

使用简单的语句

总是检验输入/输出的合法性

尽可能改进程序的效率

基本原则

- 注释
  - 注释是提高代码可读性与可维护性的关键
  - 在每个模块（构件、类、方法）前，应当用注释说明该模块的功能、算法、接口特点和数据需求
  - 在关键的部分，注释和代码的比例至少应该是1:1
  - 注释应当具有规范的格式（开发小组内商定）
    - 必要时，注释应当包含日期或版本信息

基本原则

- 简单语句
  - Simple is Beautiful

```
X=a+b; Y=c+d; Z=X+Y;
```

```
X=Y>>1;
```

```
if(!(X!=Y && Y!=Z)) …
```

基本原则

- 输入输出
  - 总检查输入和输出的合法性（依据前置/后置条件）

```
div(a,b) {
  return a/b;
}
```

```
div(a,b) {
  if(b!=0) return a/b;
  else throw …
}
```

基本原则

- 效率改进
  - 尽可能改进程序执行的效率（时间和空间复杂性）
    - 在不破坏可读性与可维护性的情况下
  - 影响效率的因素
    - 不好的设计
    - 不好的算法选择和实现
    - 不恰当的数据结构
    - 类库的使用方式、网络通信、缓存机制等

北京科技大学
University of Science and Technology Beijing

代码（软件）质量

定义

- 软件质量
  - 对明确陈述的功能和非功能需求、明确记录的开发标准以及对所有专业化软件开发应具备的隐含特征的符合度
    - 需求是软件质量测量的基础
    - 标准定义了一组用以指导软件开发方式的准则
    - 不要忘记隐式的需求

质量属性

- 功能性：软件满足需求的程度
  - 准确性、互操作性、安全性等
- 可靠性：软件可用的时间长度
  - 容错性、可恢复性等
- 易用性：易于使用的程度
  - 易学性、可操作性等

质量属性

- 效率：资源使用的优化程度
- 可维护性：软件易于修改的程度
  - 可理解性、可分析性、可修改性、可测试性等
- 可移植性：软件移植到其它环境的容易程度
  - 适应性、可安装性、可替换性等

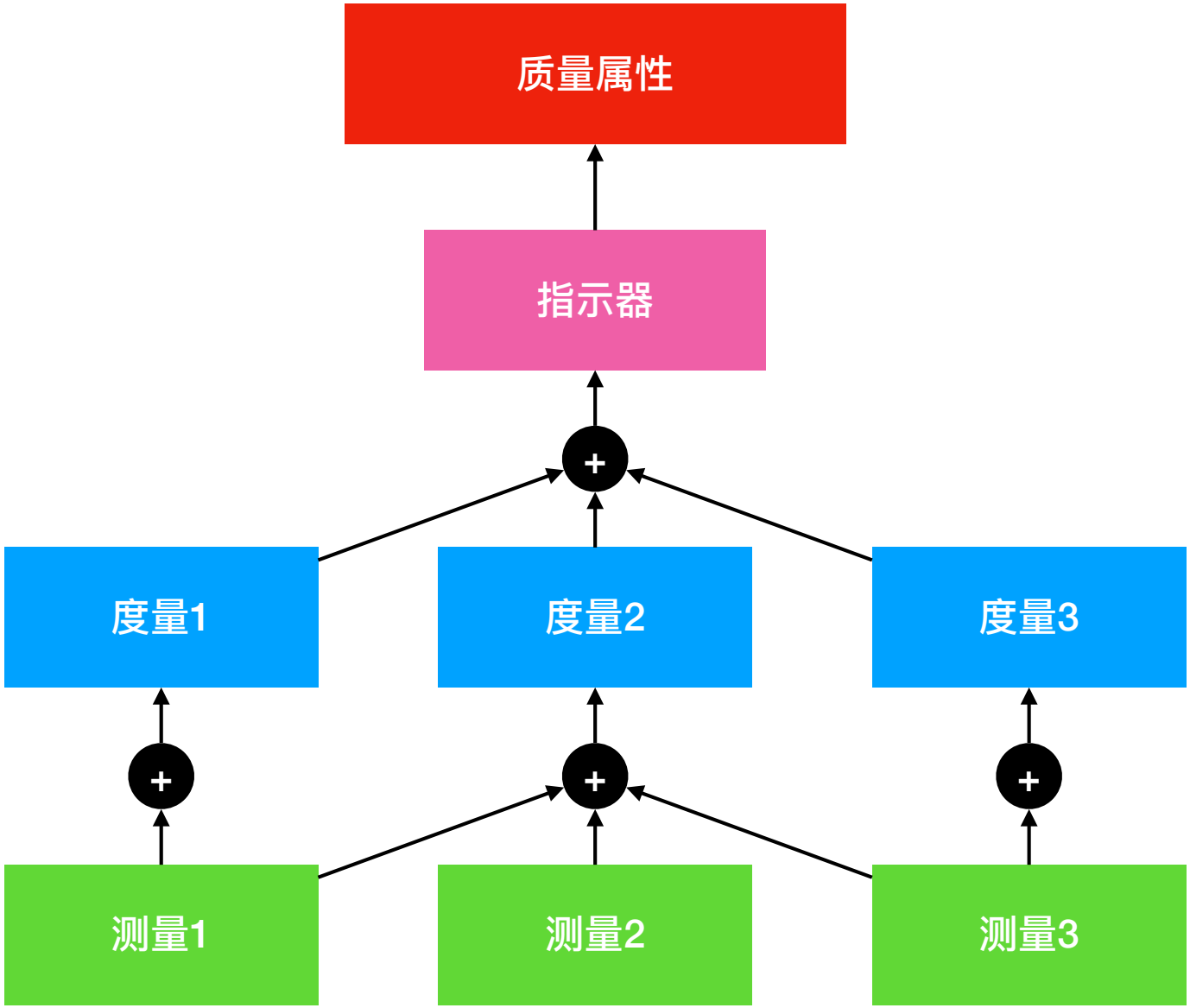
北京科技大学
University of Science and Technology Beijing

软件度量

软件度量

- 软件度量是对软件质量进行评估的常用手段

软件度量



将多个度量值进行综合，形成某个质量属性的指示器

根据统计结果进行的计算，比如正规化、统一化等

对软件制品进行的统计、计数等

软件度量

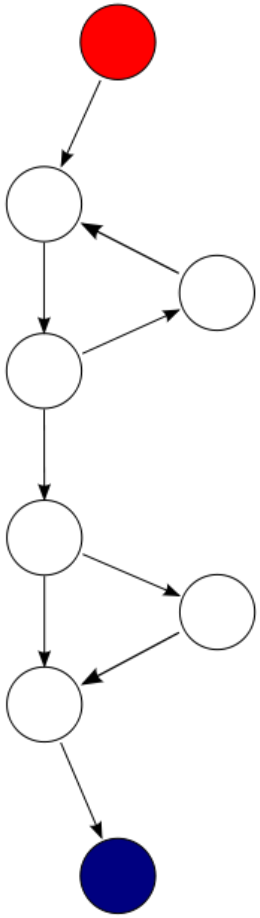
软件度量能够从某个侧面反映软件的质量属性，但软件度量是有误差的，从实验科学的角度看，度量结果存在一些有效性威胁（Threats to validity），这是软件度量学的争议所在。

软件度量

- 软件度量
  - 分析、设计、编码、测试等阶段都可以进行度量
  - 最常见的是代码度量
    - PMD：https://pmd.github.io/pmd-6.2.0/
    - findbugs：http://findbugs.sourceforge.net/findbugs2.html

基本度量方法

- Cyclomatic Complexity（环形复杂度）
  - 源程序当中（通常是一个方法中），判定的数量
  - 计算方法：π-s+2
    - π表示判断条件的数量
    - s表示出口的数量



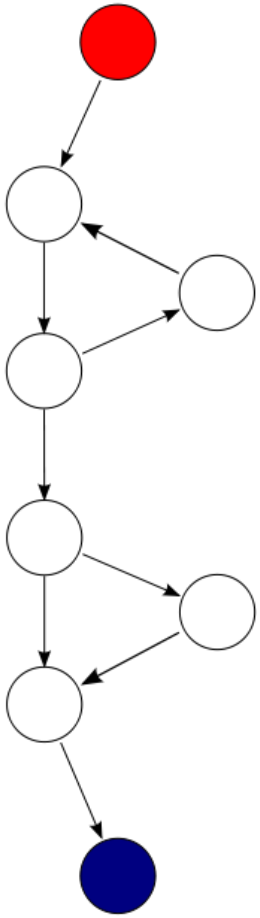
基本度量方法

```
public class Complicated {
  public void example() {
    int x = 0, y = 1, z = 2, t = 2;
    boolean a = false, b = true, c = false, d = true;
    if (a && b || b && d) {
      if (y == z) {
        x = 2;
      } else if (y == t && !d) {
        x = 2;
      } else {
        x = 2;
      }
    } else if (c && d) {
      while (z < y) {
        x = 2;
      }
    } else {
      for (int n = 0; n < t; n++) {
        x = 2;
      }
    }
  }
}
```

CYCLO=12

基本度量方法

- NPath Complexity（路径复杂度）
  - 源程序当中（通常是一个方法中），完整独立的线性路径的数量
  - 计算方法：
    - 根据语句类型和条件数量而定
      - for、while：1+条件数量
      - if：if的数量+条件的数量+无条件的else的数量（默认1）
      - ……



基本度量方法

```
public class Foo {
  public static void bar() {
   boolean a, b = true;
    try { // 2 * 2 + 2 = 6
      if (true) { // 2
        List buz = new ArrayList();
      }
     for(int i = 0; i < 19; i++) { // * 2
        List buz = new ArrayList();
      }
    } catch(Exception e) {
      if (true) { // 2
        e.printStackTrace();
      }
    }
    while (j++ < 20) { //  * 2
      List buz = new ArrayList();
    }
    switch(j) { // * 7
      case 1:
      case 2: break;
      case 3: j = 5; break;
      case 4: if (b && a) { bar(); } break;
      default: break;
    }
    do { // * 3
        List buz = new ArrayList();
    } while (a && j++ < 30);
  }
}
```

NPath=252!

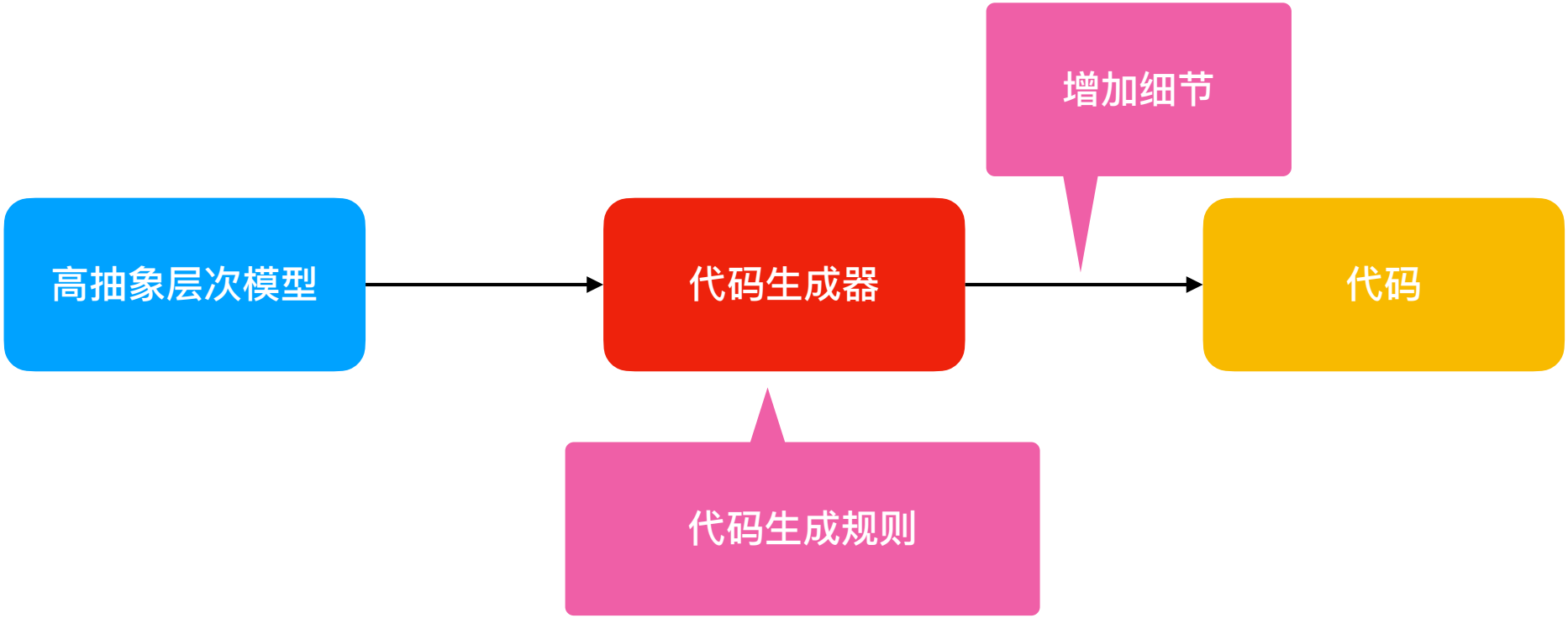
基本度量方法

- Coupling Between Objects（对象间耦合）
  - 所引入的对象（类）的数量
- God Class
  - 过于复杂的类，计算方法与类的长度、复杂度、耦合度等因素有关
- Long Method/Large Class
- ……

代码自动生成

代码自动生成

- 代码生成（code generation）是一种提高代码生产率并（有可能）改善代码质量的自动化技术



代码自动生成

```
class Person {
  private String name;
  private boolean gender;
  public String getName() {return name;}
  public boolean getGender() {return gender;}
  public void setName(String p) {name=p;}
  public void setGender(boolean p) {gender=p;}
}
```

这段代码的有效信息是什么?

抽象

```
class Person {
  String name;
  boolean gender;
}
```

在特定的语义环境下，二者等价

能否从抽象的代码（模型）中自动生成具体的代码?

代码自动生成

```
class Person {
  String name;
  boolean gender;
}
```

1.生成Person类
2.生成name属性和gender属性
3.生成name属性的getter和setter方法
4.生成gender属性的getter和setter方法

如何生成？

```
class Person {
  private String name;
  private boolean gender;
  public String getName() {return name;}
  public boolean getGender() {return gender;}
  public void setName(String p) {name=p;}
  public void setGender(boolean p) {gender=p;}
}
```

代码自动生成

```
class ⟨C⟩ {
  ⟨T1⟩ ⟨A1⟩;
  ....
  ⟨Tn⟩ ⟨An⟩;
}
```

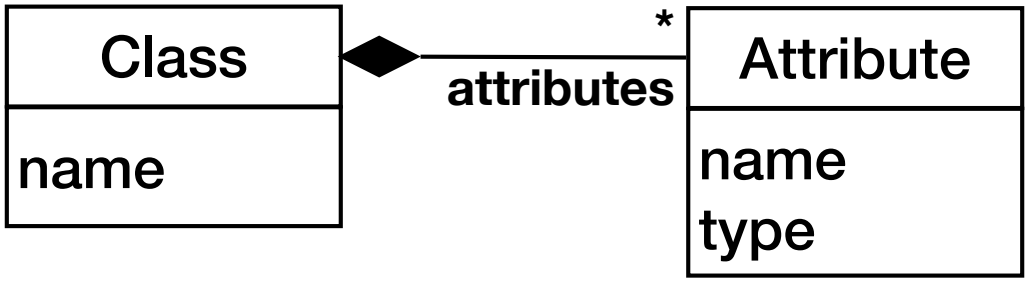
进一步抽象和一般化

```
class ⟨C⟩ {
  private ⟨T1⟩ ⟨A1⟩;
  …
  private ⟨Tn⟩ ⟨An⟩;
  public ⟨T1⟩ get⟨A1⟩(){return  ⟨A1⟩;}
  …
  public ⟨Tn⟩ get⟨An⟩(){return  ⟨An⟩;}
  public void set⟨A1⟩(⟨T1⟩ p){⟨A1⟩=p;}
  …
  public void set⟨An⟩(⟨Tn⟩ p){⟨An⟩=p;}
}
```

代码自动生成

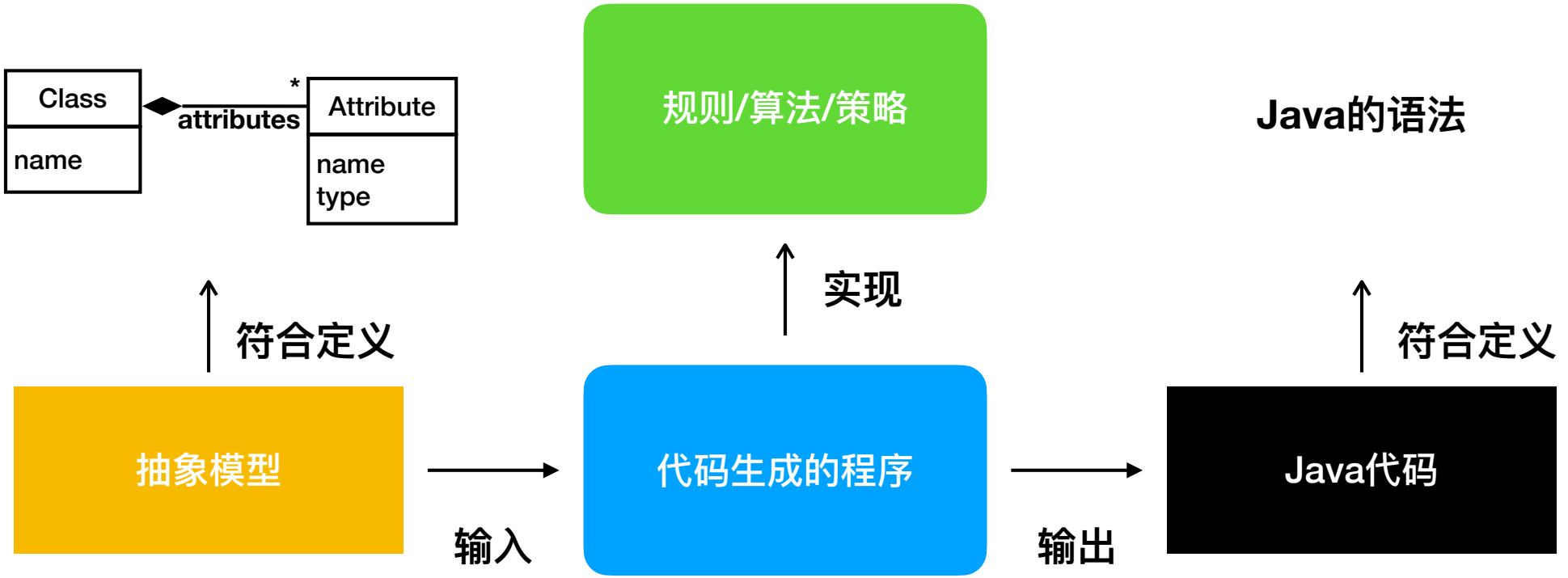
```
class ‹C› {
  ‹T1› ‹A1›;
  ....
  ‹Tn› ‹An›;
}
```

定义抽象模型的结构



代码自动生成

定义代码生成规则/算法/策略/程序



代码自动生成

```
Input: a class definition
Output: a piece of code that refines the class definition

cls = the class definition
generate a Java class, whose name is cls.name
foreach Attribute a in cls.ownedAttributes
  generate an attribute in the Java class
  generate a getter method in the Java class
  generate a setter method in the Java class
end
```

Higher-order programming!

代码自动生成

- 基于模板的代码生成
  - JET: http://www.eclipse.org/modeling/m2t/?project=jet
  - Acceleo: https://www.eclipse.org/acceleo
  - Xtend: http://www.eclipse.org/xtend/
- 基于规则的代码生成
- 基于深度学习的代码生成

代码自动生成

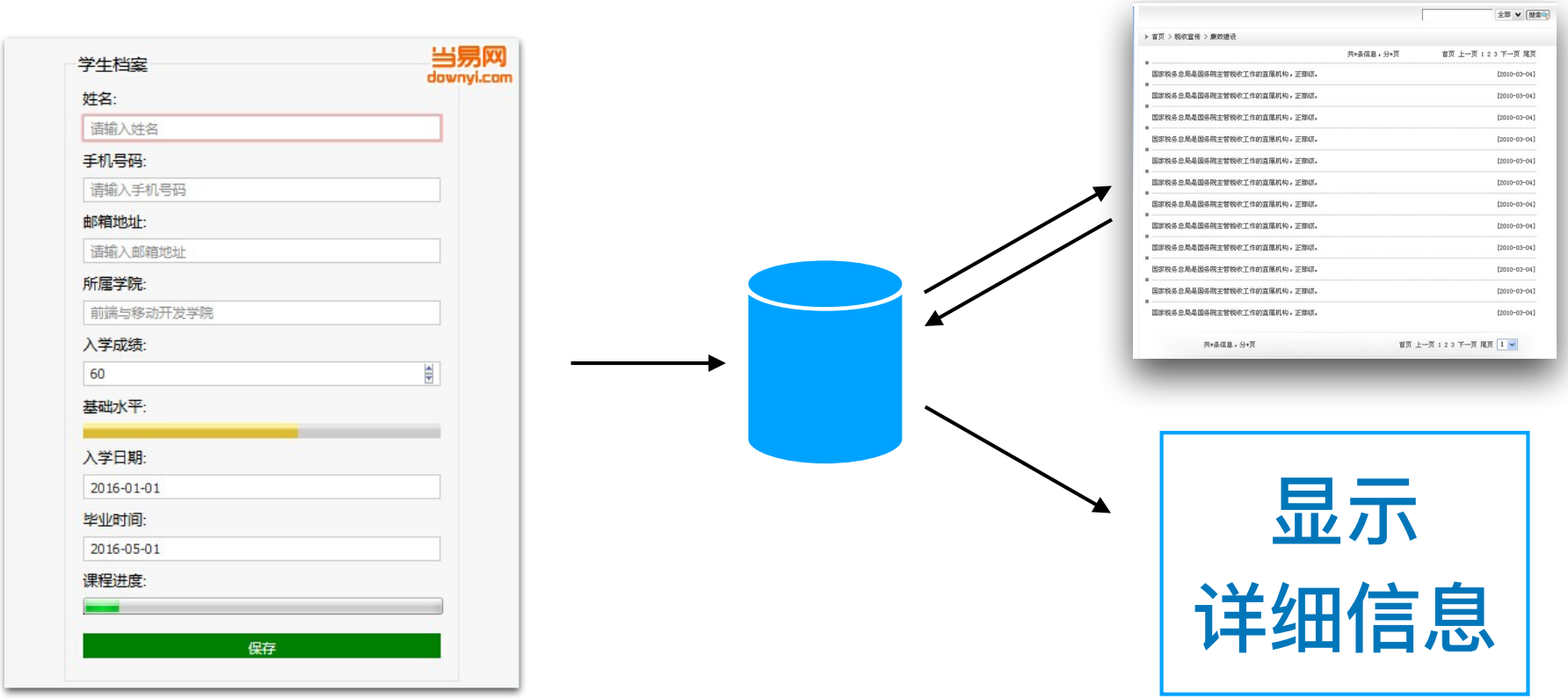
类似JSP的语法

```
public class <%=cls.name%> {

  <%for(Attribute a : cls.ownedAttributes) { %>
    private <%=a.type%> <%=a.name%>;
    public <%=a.type%> get<%=a.name.upperFirst()%>() {
      return this.<%=a.name%>;
    }
    public void set<%=a.name.upperFirst()%>(<%=a.type%> value) {
      this.<%=a.name%> = value;
    }
  <%}%>
}
```

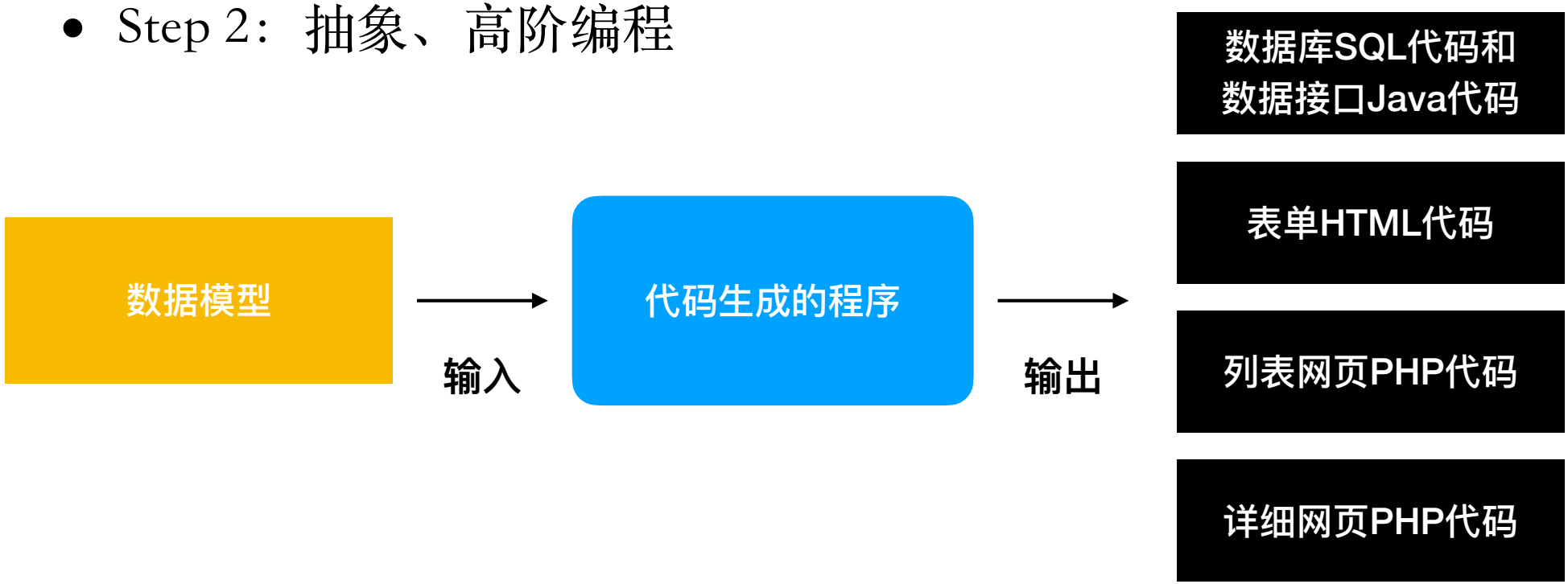
代码自动生成

- 例子：
  - Web应用中经常需要填写各种表单；提交数据后保存在后台数据库；然后可以通过列表的形式展示出所有记录，并点击其中一条显示详细内容。

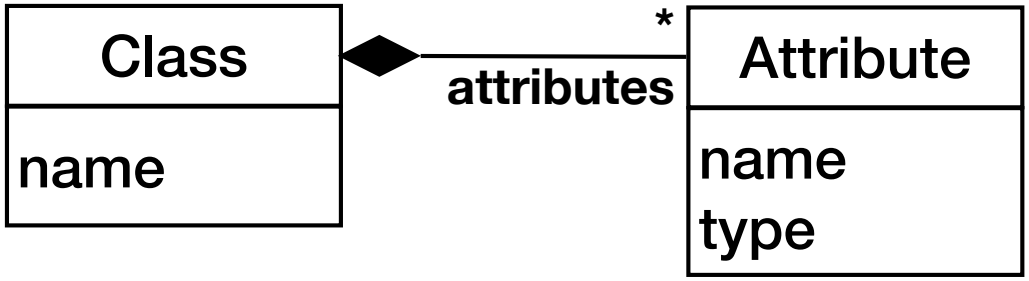


代码自动生成

- Step 1：共性-变化性分析
  - 整体过程是相同的，提交的数据定义不同
- Step 2：抽象、高阶编程



代码自动生成



产生SQL命令：

```
foreach Class c in Input
  generate a “create” table SQL statement, where the table name is c.name
    foreach Attribute a in c
      append a column to the “create” statement, whose name and type are a.name and a.type
    end
end
```

代码自动生成

```
create table <%=c.name%>
  (
  <%for(Attribute a : cls.ownedAttributes) {
    if(cls.ownedAttributes.first()==a) {%>,
    <%}%> <%=a.name%> <%=a.type%>
  <%}%>
  )

select * from <%=c.name%>
```

代码自动生成

- 提示
  - 开发代码生成器比直接开发相应代码更加复杂
    - 但可以反复使用，从而提高生产率
    - 尽可能减少对自动产生代码的维护（需要好的设计）
  - 代码生成的核心是共性变化性分析
  - 代码生成的难点是定义抽象模型（即代码生成器的输入）
  - 代码生成器不是万能的

北京科技大学
University of Science and Technology Beijing

扩展阅读

智能化软件开发微访谈·第十六期：低代码/无代码开发

原创 CodeWisdom CodeWisdom 1周前

CodeWisdom

“智能化软件开发沙龙是由CodeWisdom团队组织的围绕智能化软件开发、数据驱动的软件开发质量与效能分析、云原生与智能化运维等相关话题开展的线上沙龙，通过微信群访谈交流等线上交流方式将学术界与工业界专家学者汇聚起来，共同分享前沿研究进展与业界实践，共同探讨未来技术发展方向。”

https://mp.weixin.qq.com/s/Wdzwyv70AlmGore0jdaXrw

北京科技大学
University of Science and Technology Beijing

第四次作业

为Aquila增加代码生成的能力

请为Aquila类图实现代码生成功能，能够将任意一个Aquila类图模型（输入）转化成Java代码（或其他编程语言）