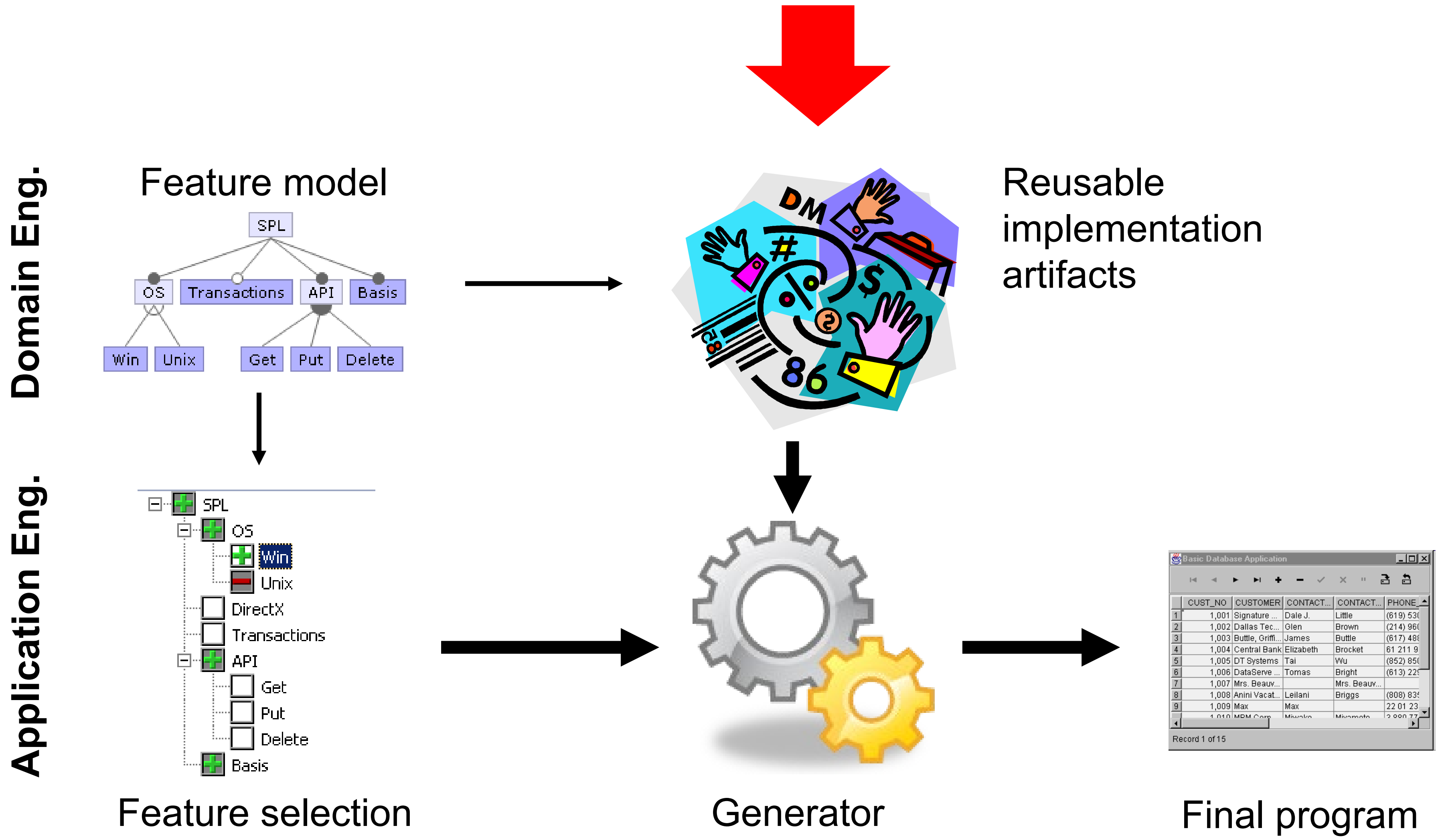
Software Product Lines
Part 3: Runtime Variability

Daniel Strüber, Radboud University

with courtesy of: **Sven Apel, Christian Kästner, Gunter Saake**

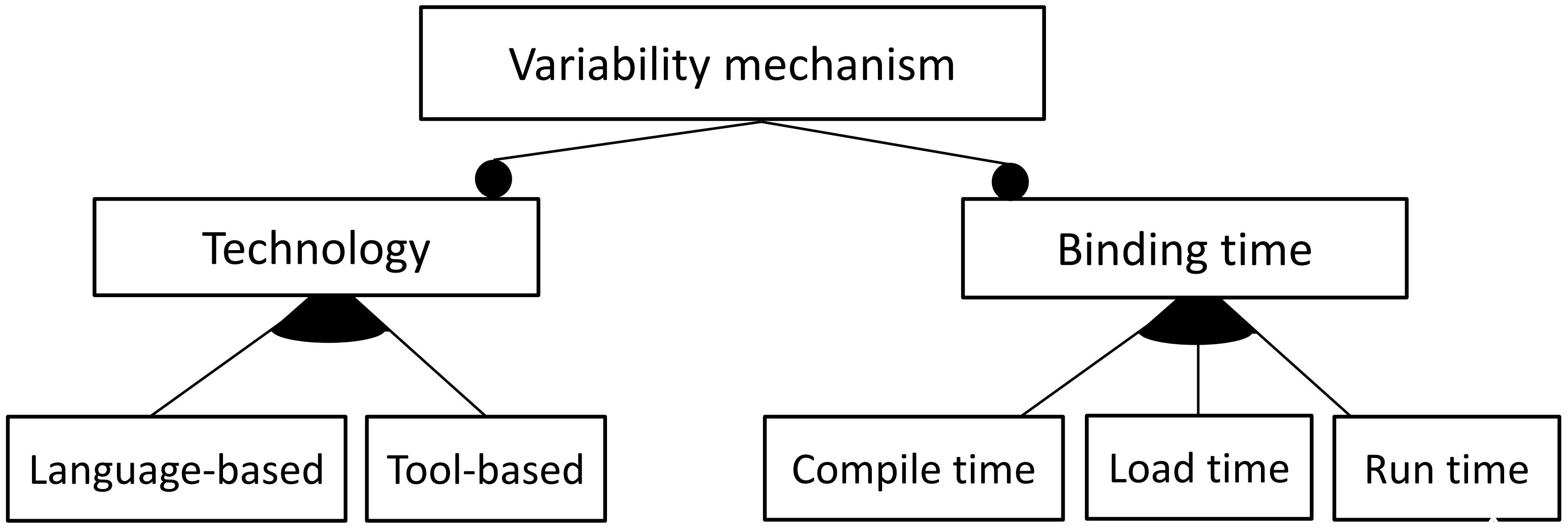
How to implement variability?



	CUST_NO	CUSTOMER	CONTACT...	CONTACT...	PHONE_
1	1,001	Signature ...	Dale J.	Little	(619) 53
2	1,002	Dallas Tec...	Glen	Brown	(214) 96
3	1,003	Buttle, Griffi...	James	Buttle	(617) 48
4	1,004	Central Bank	Elizabeth	Brocket	61 211 9
5	1,005	DT Systems	Tai	Wu	(852) 85
6	1,006	DataServe ...	Tomas	Bright	(613) 22
7	1,007	Mrs. Beauv...		Mrs. Beauv...	
8	1,008	Anini Vacat...	Leilani	Briggs	(808) 83
9	1,009	Max	Max		22 01 23

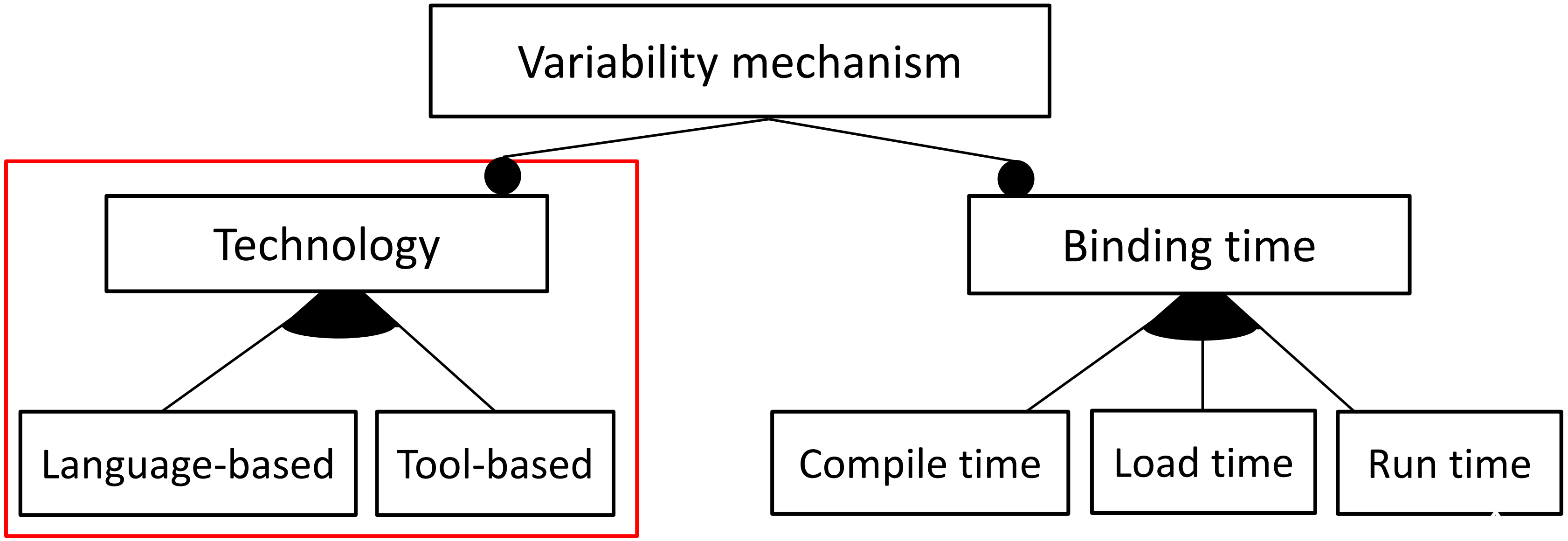
Record 1 of 15

Variability mechanisms: a broad categorization



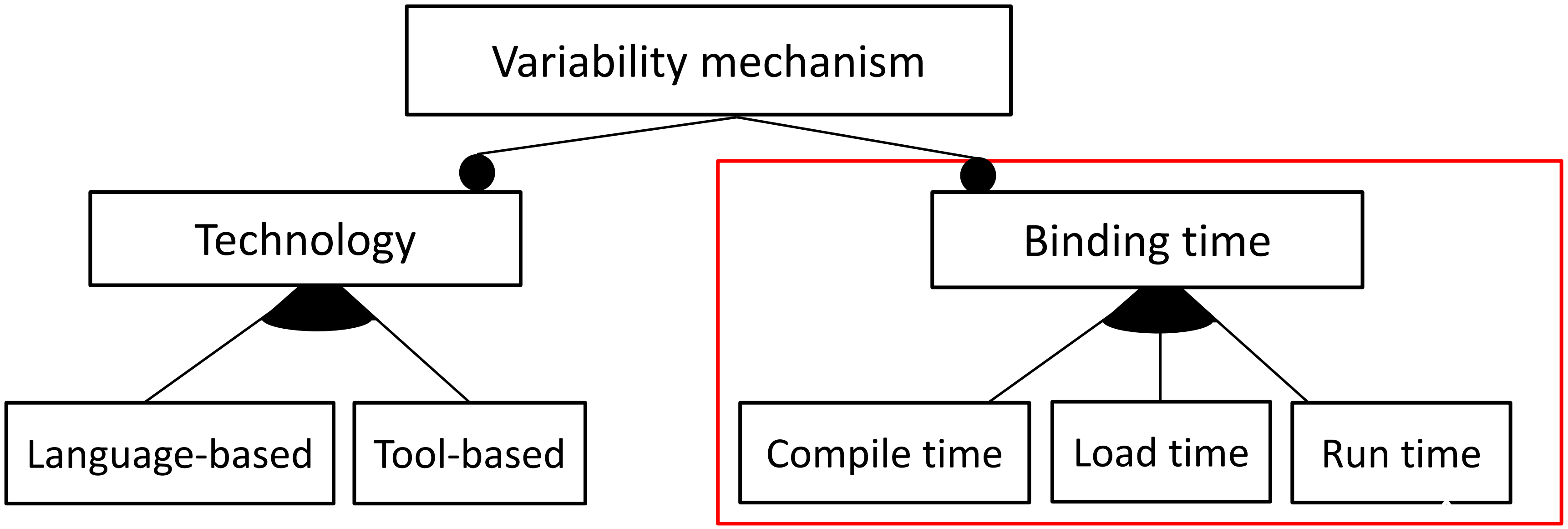
Approaches to implementing variability

- **Language-based**: Implementation + product generation based on mechanisms of the programming language
- **Tool-based**: Use external tools to establish connection between features and code and to generate products



Approaches to implementing variability

- **Compile time**: Feature selection + product generation before/during compilation; only relevant code included
- **Load time**: Compiled program supports all products; feature selection when program is started
- **Run time**: Compiled program supports all products; feature selection may change during execution (dynamic reconfiguration)



Agenda

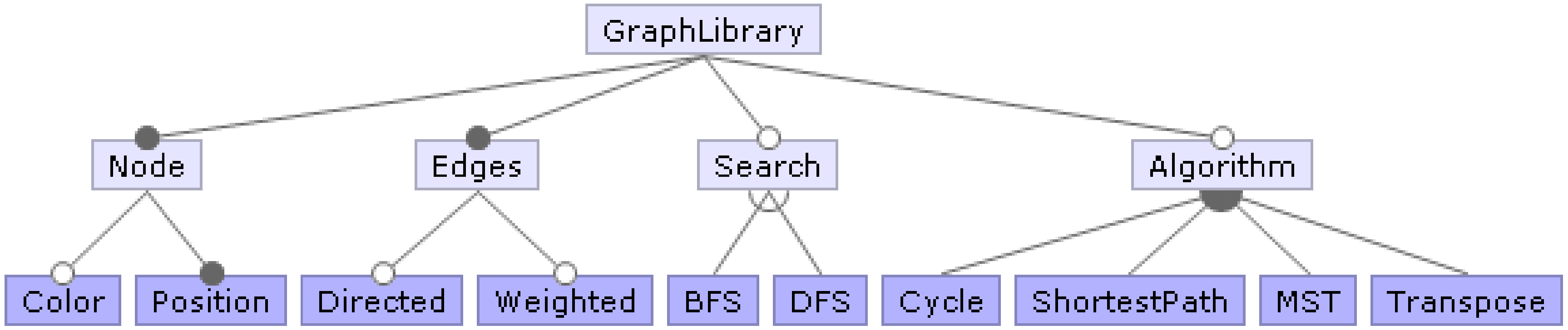
- Graph example
- Variability mechanism 1: runtime parameters
- Refresher: Modularity
- Variability mechanism 2: variablity with design patterns
- Limitations of available mechanisms

An example

Example: Graph library

- Will be a running example from here (like chat system in assignments)
- Library of graph structures and algorithms
  - weighted vs. unweighted edges
  - directed vs. undirected edges
  - colored nodes
  - algorithms: shortest path, minimal spanning tree, transitive closure…

Graph feature model



Implementation (without variability)

```
class Graph {
  List nodes = new ArrayList(); List edges = new ArrayList();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

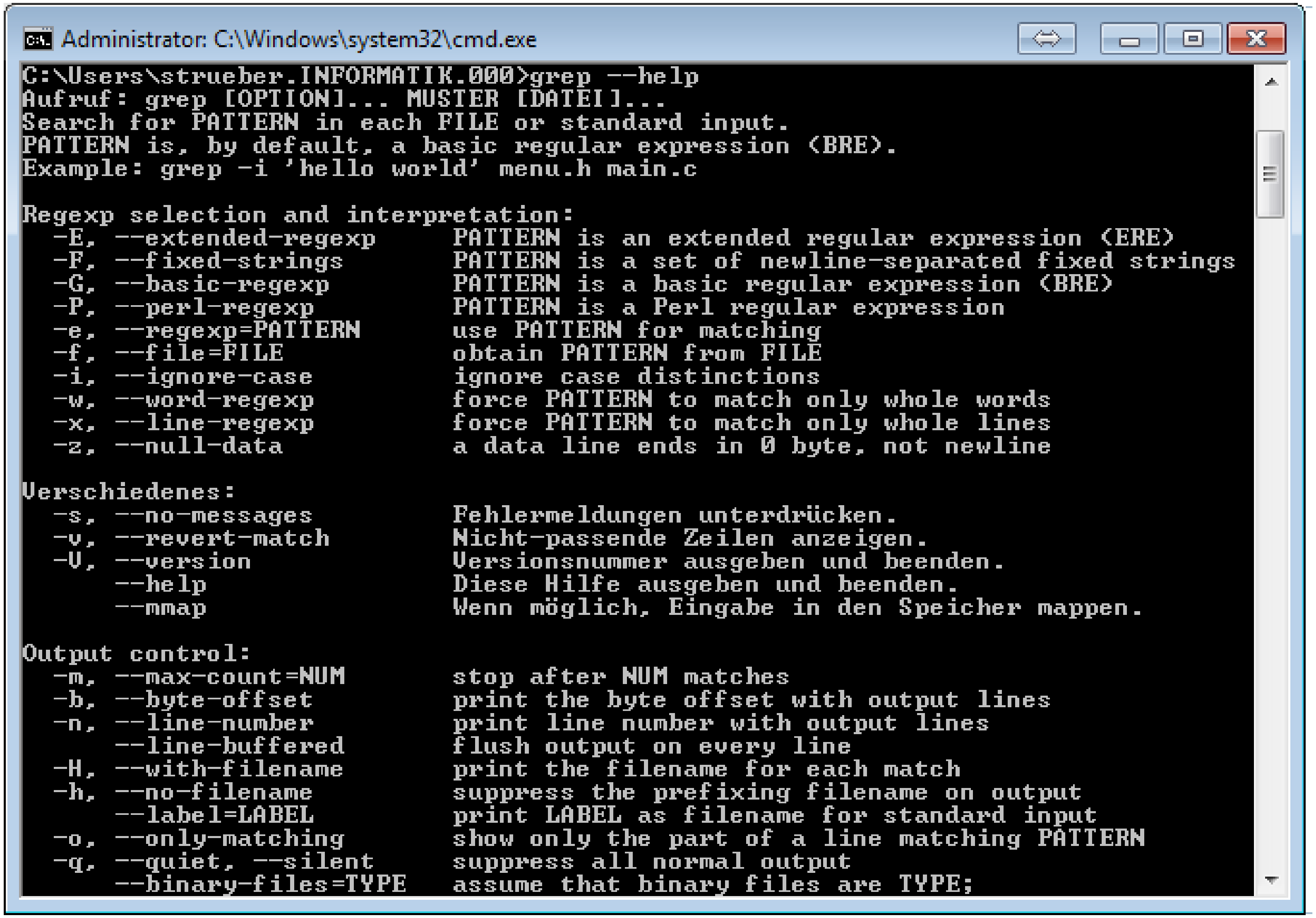
```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
    weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

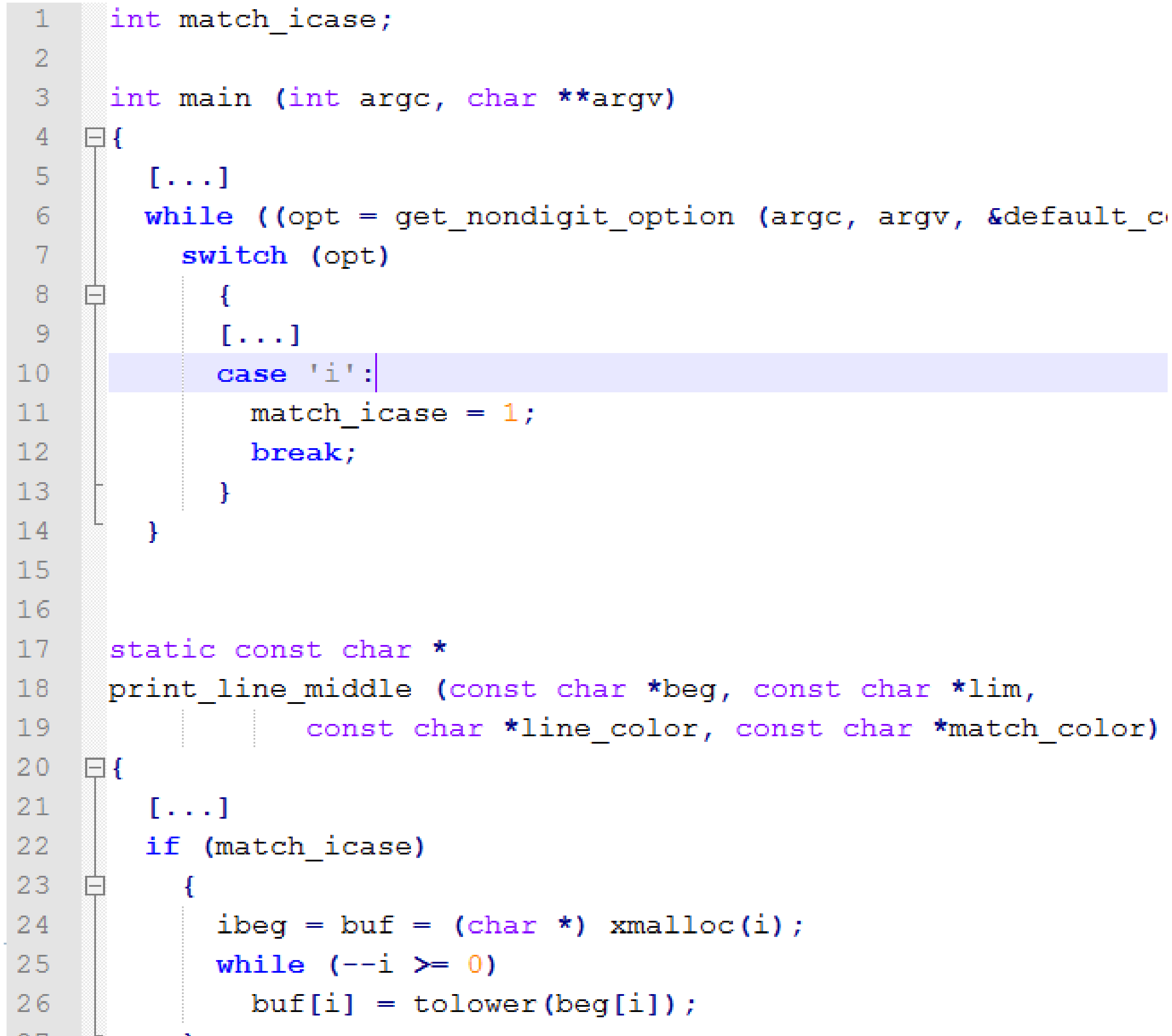
Runtime parameters

Parameter



```
C:\Users\strueber.INFORMATIK.000>grep --help
Aufruf: grep [OPTION]... MUSTER [DATEI]...
Search for PATTERN in each FILE or standard input.
PATTERN is, by default, a basic regular expression (BRE).
Example: grep -i 'hello world' menu.h main.c

Regexp selection and interpretation:
  -E, --extended-regexp     PATTERN is an extended regular expression (ERE)
  -F, --fixed-strings       PATTERN is a set of newline-separated fixed strings
  -G, --basic-regexp        PATTERN is a basic regular expression (BRE)
  -P, --perl-regexp         PATTERN is a Perl regular expression
  -e, --regexp=PATTERN      use PATTERN for matching
  -f, --file=FILE           obtain PATTERN from FILE
  -i, --ignore-case         ignore case distinctions
  -w, --word-regexp         force PATTERN to match only whole words
  -x, --line-regexp         force PATTERN to match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline

Verschiedenes:
  -s, --no-messages         Fehlermeldungen unterdrücken.
  -v, --revert-match        Nicht-passende Zeilen anzeigen.
  -V, --version             Versionsnummer ausgeben und beenden.
      --help                Diese Hilfe ausgeben und beenden.
      --mmap                Wenn möglich, Eingabe in den Speicher mappen.

Output control:
  -m, --max-count=NUM       stop after NUM matches
  -b, --byte-offset         print the byte offset with output lines
  -n, --line-number         print line number with output lines
      --line-buffered       flush output on every line
  -H, --with-filename       print the filename for each match
  -h, --no-filename         suppress the prefixing filename on output
      --label=LABEL         print LABEL as filename for standard input
  -o, --only-matching       show only the part of a line matching PATTERN
  -q, --quiet, --silent     suppress all normal output
      --binary-files=TYPE   assume that binary files are TYPE;
```

Parameter –i in grep



```
int match_icase;

int main (int argc, char **argv)
{
  [...]
  while ((opt = get_nondigit_option (argc, argv, &default_c
    switch (opt)
      {
      [...]
      case 'i':
        match_icase = 1;
        break;
      }
  }


static const char *
print_line_middle (const char *beg, const char *lim,
                   const char *line_color, const char *match_color)
{
  [...]
  if (match_icase)
    {
      ibeg = buf = (char *) xmalloc(i);
      while (--i >= 0)
        buf[i] = tolower(beg[i]);
```

Global configuration options

```
class Conf {
   public static boolean Logging = false;
   public static boolean Windows = false;
   public static boolean Linux = true;
}
class Main {
  public void foo() {
     if (Conf.Logging)
        log(„running foo()“);
     if (Conf.Windows)
        callWindowsMethod();
     else if (Conf.Linux)
        callLinuxMethod();
     else
        throw RuntimeException();
  }
}
```

Implementation

```
class Conf {
  public static boolean COLORED = true;
  public static boolean WEIGHTED = false;
}
```

```
class Graph {
  List nodes = new ArrayList(); List edges = new ArrayList();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
    }
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf.COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

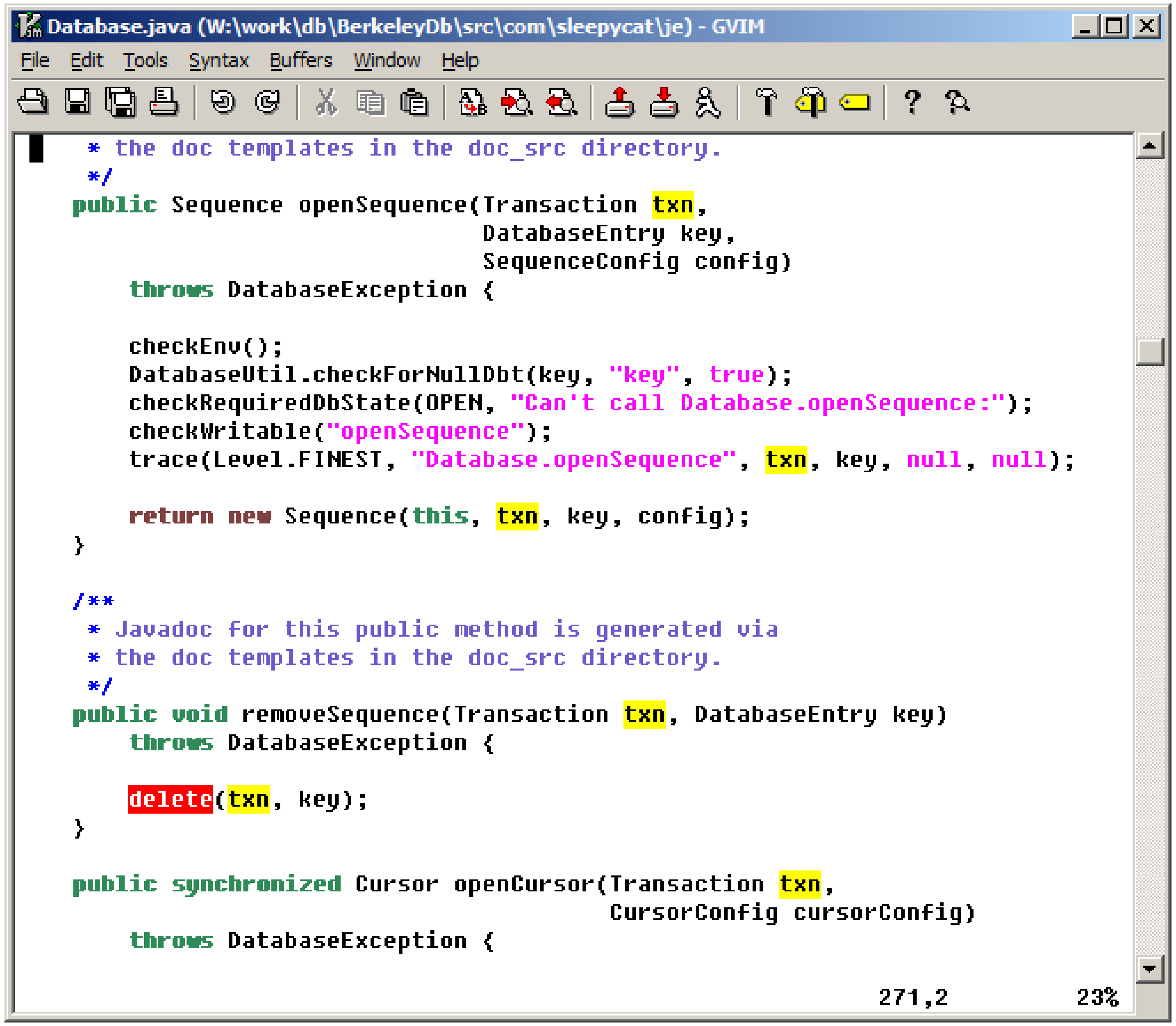
```
class Weight { void print() { ... } }
```

Parameter passing

avoid global variables.

instead:
pass parameters
though methods

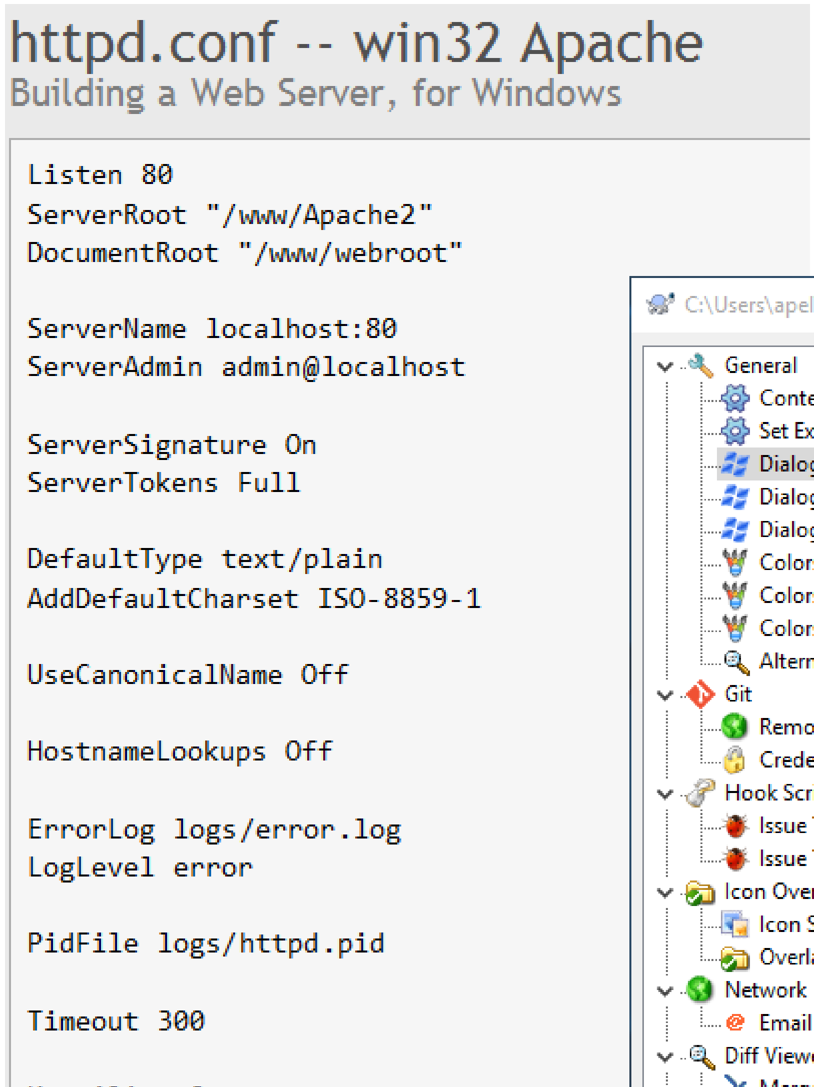
(can drag on through
many methods...)


Database.java (W:\work\db\BerkeleyDb\src\com\sleepycat\je) - GVIM
File Edit Tools Syntax Buffers Window Help

```
     * the doc templates in the doc_src directory.
     */
    public Sequence openSequence(Transaction txn,
                                 DatabaseEntry key,
                                 SequenceConfig config)
        throws DatabaseException {

        checkEnv();
        DatabaseUtil.checkForNullDbt(key, "key", true);
        checkRequiredDbState(OPEN, "Can't call Database.openSequence:");
        checkWritable("openSequence");
        trace(Level.FINEST, "Database.openSequence", txn, key, null, null);

        return new Sequence(this, txn, key, config);
    }

    /**
     * Javadoc for this public method is generated via
     * the doc templates in the doc_src directory.
     */
    public void removeSequence(Transaction txn, DatabaseEntry key)
        throws DatabaseException {

        delete(txn, key);
    }

    public synchronized Cursor openCursor(Transaction txn,
                                          CursorConfig cursorConfig)
        throws DatabaseException {
                                                        271,2          23%
```

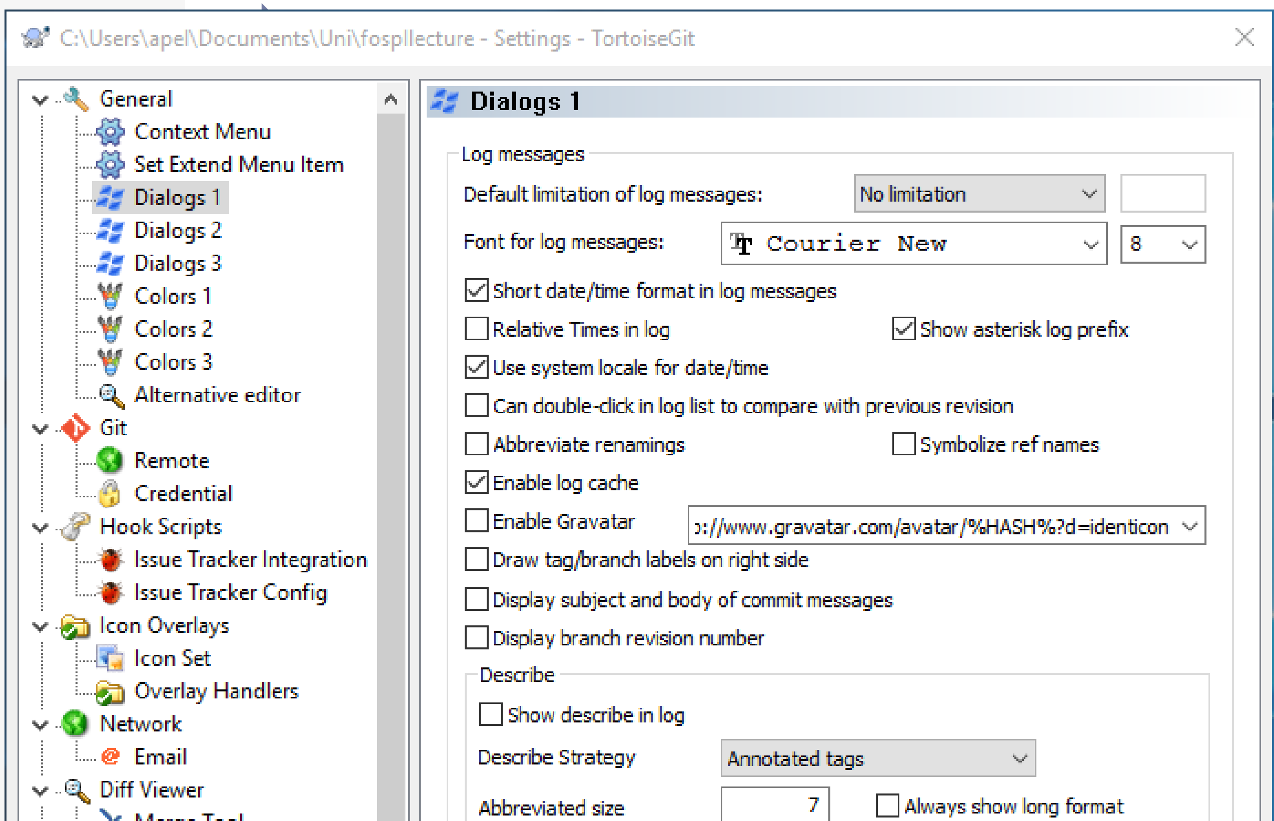
Configuration

- command line parameters
- config file
- dialog
- source code
- registry



httpd.conf -- win32 Apache
Building a Web Server, for Windows

```
Listen 80
ServerRoot "/www/Apache2"
DocumentRoot "/www/webroot"

ServerName localhost:80
ServerAdmin admin@localhost

ServerSignature On
ServerTokens Full

DefaultType text/plain
AddDefaultCharset ISO-8859-1

UseCanonicalName Off

HostnameLookups Off

ErrorLog logs/error.log
LogLevel error

PidFile logs/httpd.pid

Timeout 300
```



Dialogs 1

Log messages

Default limitation of log messages: No limitation

Font for log messages: Courier New 8

- [x] Short date/time format in log messages
- [ ] Relative Times in log
- [x] Show asterisk log prefix
- [x] Use system locale for date/time
- [ ] Can double-click in log list to compare with previous revision
- [ ] Abbreviate renamings
- [ ] Symbolize ref names
- [x] Enable log cache
- [ ] Enable Gravatar ɔ://www.gravatar.com/avatar/%HASH%?d=identicon
- [ ] Draw tag/branch labels on right side
- [ ] Display subject and body of commit messages
- [ ] Display branch revision number

Describe

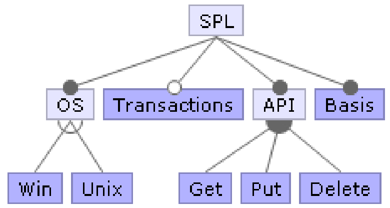
- [ ] Show describe in log

Describe Strategy: Annotated tags

Abbreviated size: 7
- [ ] Always show long format

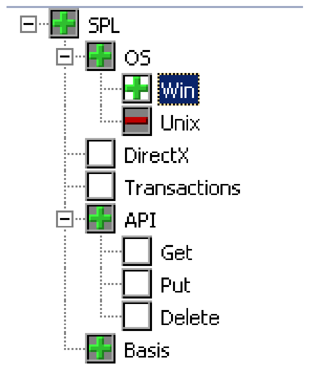
Domain Eng.

parameter list
(feature model)



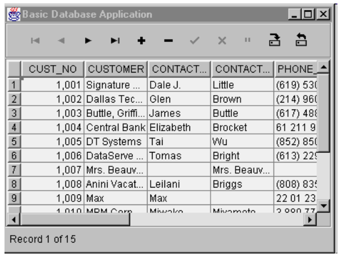
program with
runtime parameters

Application Eng.



parameter selection
(feature selection)

setting of parameters



Basic Database Application

	CUST_NO	CUSTOMER	CONTACT...	CONTACT...	PHONE_
1	1,001	Signature ...	Dale J.	Little	(619) 53
2	1,002	Dallas Tec...	Glen	Brown	(214) 96
3	1,003	Buttle, Griffi...	James	Buttle	(617) 48
4	1,004	Central Bank	Elizabeth	Brocket	61 211 9
5	1,005	DT Systems	Tai	Wu	(852) 85
6	1,006	DataServe ...	Tomas	Bright	(613) 22
7	1,007	Mrs. Beauv...		Mrs. Beauv...	
8	1,008	Anini Vacat...	Leilani	Briggs	(808) 83
9	1,009	Max	Max		22 01 23
	1,010	MBM Corp	Miwako	Miyamoto	3 880 77

Record 1 of 15

program execution
with desired behavior

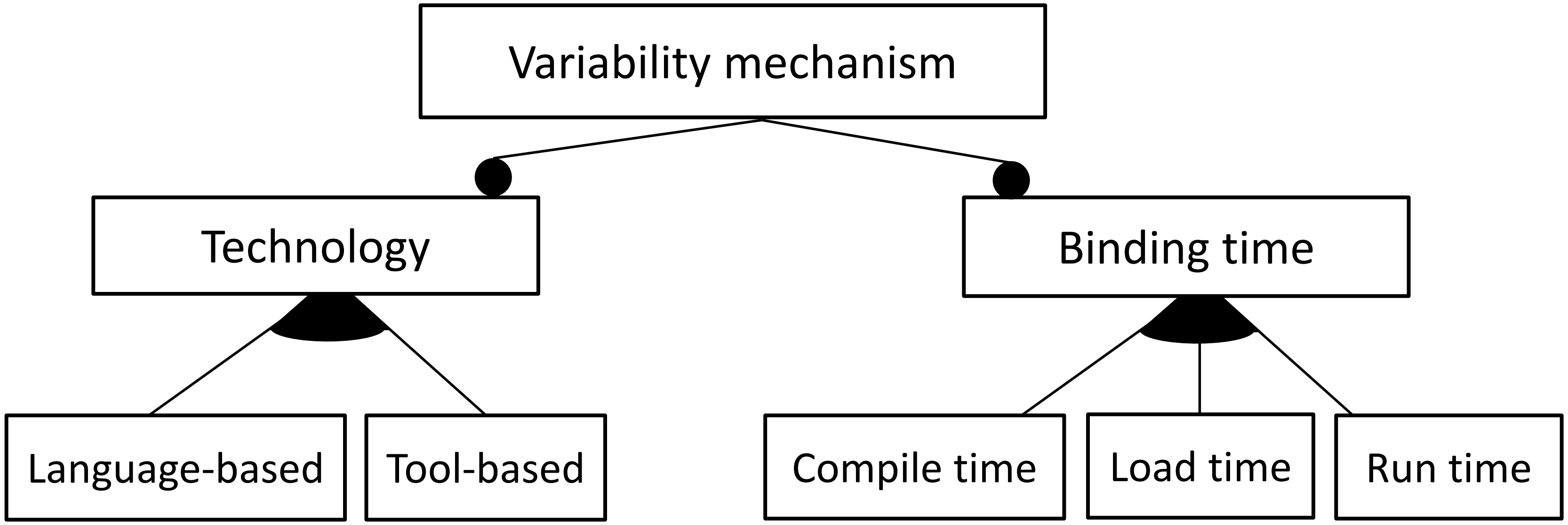
Discussion

- Simple and widely used
- Variability spread in entire program
- Global variables vs. long parameter lists

- Checking of configuration?
- Changes at runtime possible?
- Protection against use of deactivated functionality?

- No generator; always full set of variants deployed
  - code size; memory use, runtime performance
  - unused functionality as risk factor

Zoom quiz: parameters?



Refresher: Modularity

What is modularity?

- Modularity = encapsulation and cohesion
- Encapsulation: hide implementation details behind an interface
- Cohesion: group related program constructs in a single addressable unit (for example, packages, classes…)
- Encapsulated and cohesive units can be read, understood and changed in isolation
- Reduces complexity during software engineering lifecycle

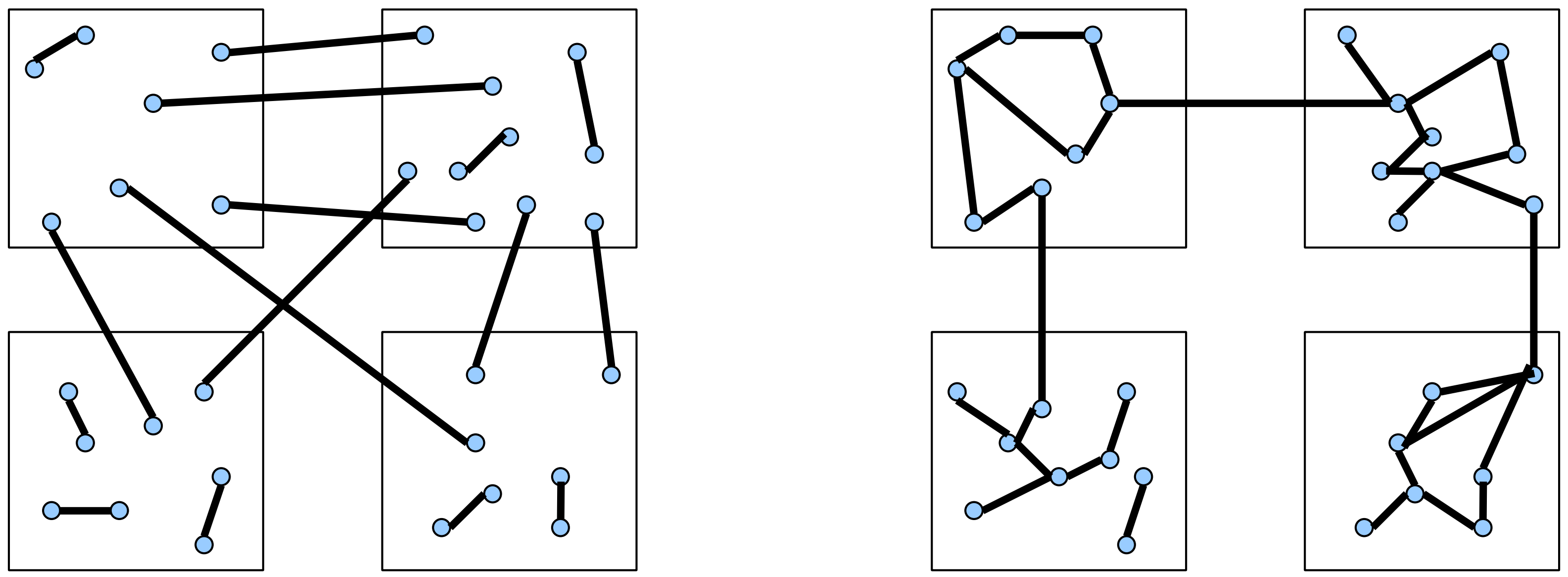
Encapsulation

```
public class ArrayList<E> {
   public void add(int index, E element) {
      if (index > size || index < 0)
         throw new IndexOutOfBoundsException(
            "Index: "+index+", Size: "+size);
      ensureCapacity(size+1);
      System.arraycopy(elementData, index,
         elementData, index + 1, size - index);
      elementData[index] = element;
      size++;
   }
   public int indexOf(Object o) {
      if (o == null) {
         for (int i = 0; i < size; i++)
            if (elementData[i]==null)
            return i;
      } else {
         for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
            return i;
      }
      return -1;
    }
   .......
}
```

```
public interface List<E> {
   void add(int index, E element);
   int indexOf(Object o);
   ....
}
```

- Implementation details are hidden
- Interface describes behavior
- Implementation becomes interchangable

Cohesion/coupling – Example



- Grouping of methods/tasks
- Many calls across group boundaries
- Group implements different concerns

Why modularity?

- Software becomes easier to read and understand (*divide and conquer*)
- Hide complexity of parts behind interfaces (*information hiding*)
- Easier to maintain, changes happen locally (*maintainability*)
- Parts of the software can be reused (*reusability*)
- Modules can also be composed in a different way in new contexts (*variability*)

Problems of runtime parameters? – Scattered code

Code Scattering

```
class Graph {
  List nodes = new ArrayList(); L
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
    }
  }
}
```

```
class Node {
                    w Color();

    if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Problems of runtime parameters? – Tangled Code

```
class Graph {
  List nodes = new ArrayList(); List edges = new ArrayList();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
  Edge add(Node n, Node m, Weight w)
    if (!Conf.WEIGHTED) throw RuntimeException();
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = w; return e;
  }
  void print() {
  }
}
```

Code Tangling

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

Problems of runtime parameters? – Replicated Code

```
class Graph {
  List nodes = new ArrayList(); List edges = new ArrayList();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n); nodes.add(m); edges.add(e);
    if (Conf.WEIGHTED) e.weight = new Weight();
    return e;
  }
    nodes.add(n); nodes.add(m); edges.add(e);
    e.weight = w; return e;
  }
  void print() {
    for(int i = 0; i < edges.size(); i++) {
      ((Edge)edges.get(i)).print();
    }
  }
}
```

Code Replication

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
   if (Conf.COLORED) Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Weight weight;
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    if (Conf. COLORED) Color.setDisplayColor(color);
    a.print(); b.print();
    if (!Conf.WEIGHTED)  weight.print();
  }
}
```

```
class Color {
  static void setDisplayColor(Color c) { ... }
}
```

```
class Weight { void print() { ... } }
```

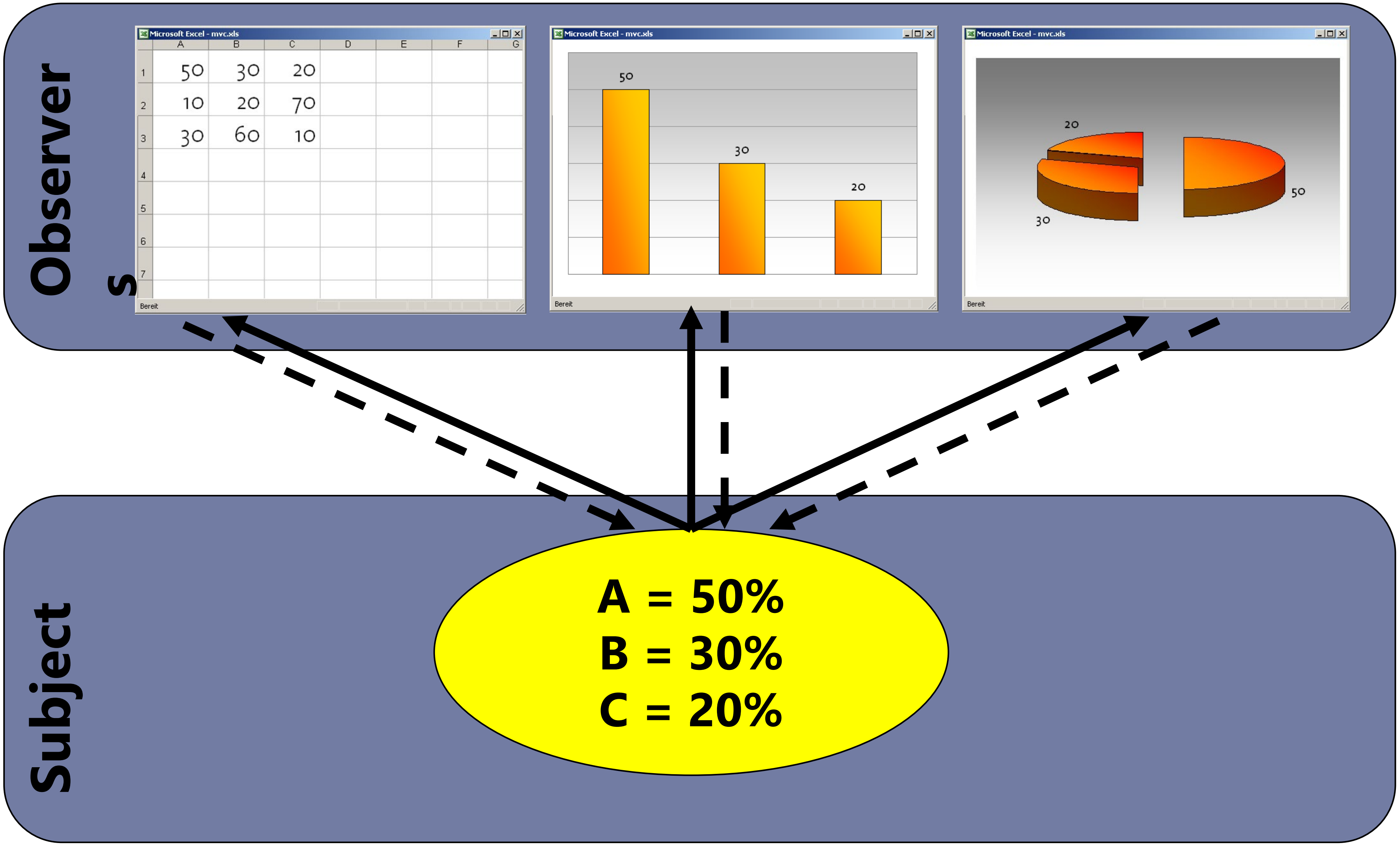
Design Patterns for variability

Design Patterns

- Patterns for design of solutions for recurring problems
- Many design patters exist for variability, decoupling and extendibility

- We consider a selection:
  - Observer
  - Template Method
  - Strategy
  - Decorator

Observer Pattern

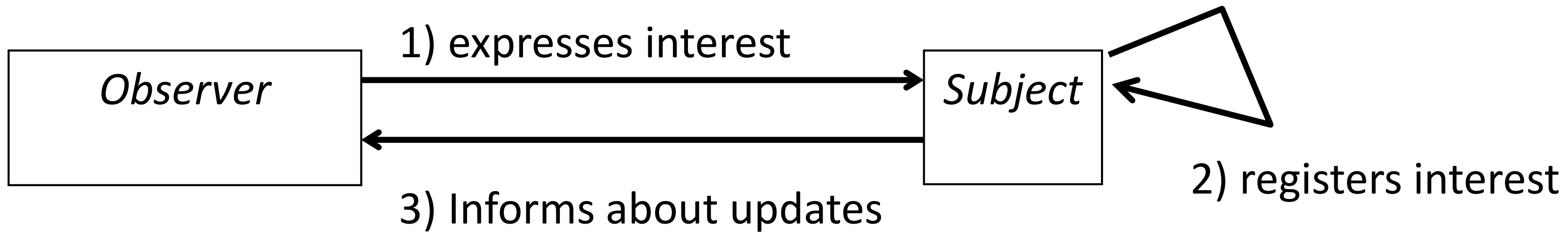


	A	B	C
1	50	30	20
2	10	20	70
3	30	60	10

[source: Meyer/Bay]

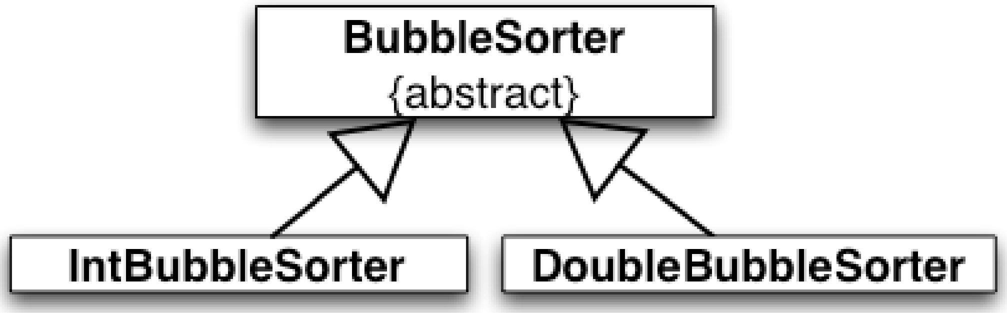
Observer Pattern

“Define[*s*] *a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”* [GoF, p 293]



- In implementation
  - Class or interface for observer (interface adds flexibility)
  - Class for subject
  - Subject maintains list of observers
  - Subject.addToObservers(Observer)   (called by observer)
  - Observer.notify()   (called by subject)

Template Method Pattern

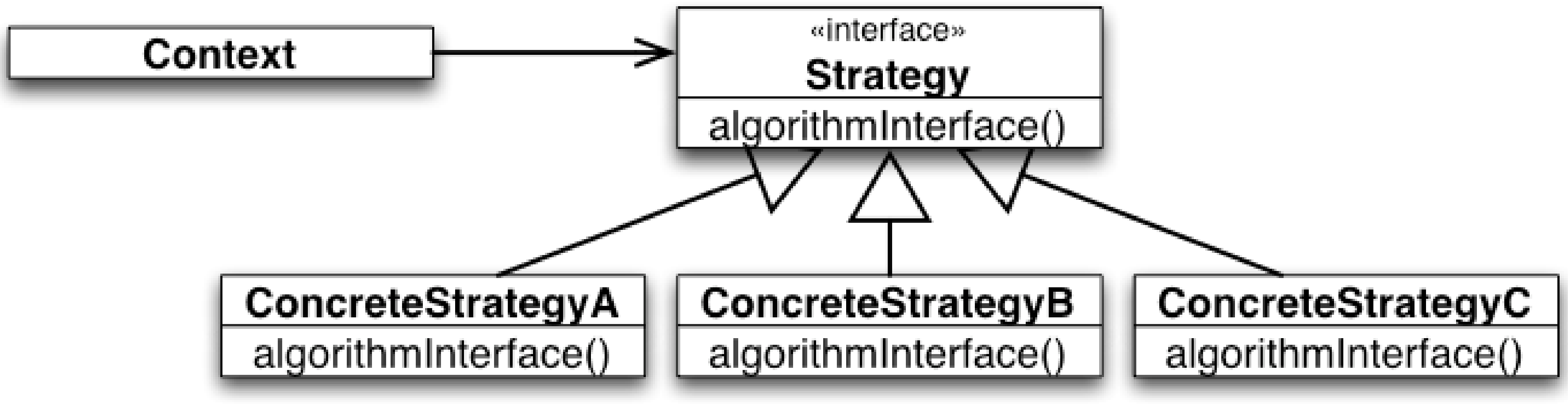


```
public abstract class BubbleSorter{
        protected int length = 0;
        protected void sort() {
                if (length <= 1) return;
                for (int nextToLast= length-2;
                        nextToLast>= 0; nextToLast--)
                        for (int index = 0;
                                index <= nextToLast; index++)
                                if (outOfOrder(index)) swap(index);
        }
        protected abstract void swap(int index);
        protected abstract boolean outOfOrder(int index);
}
```

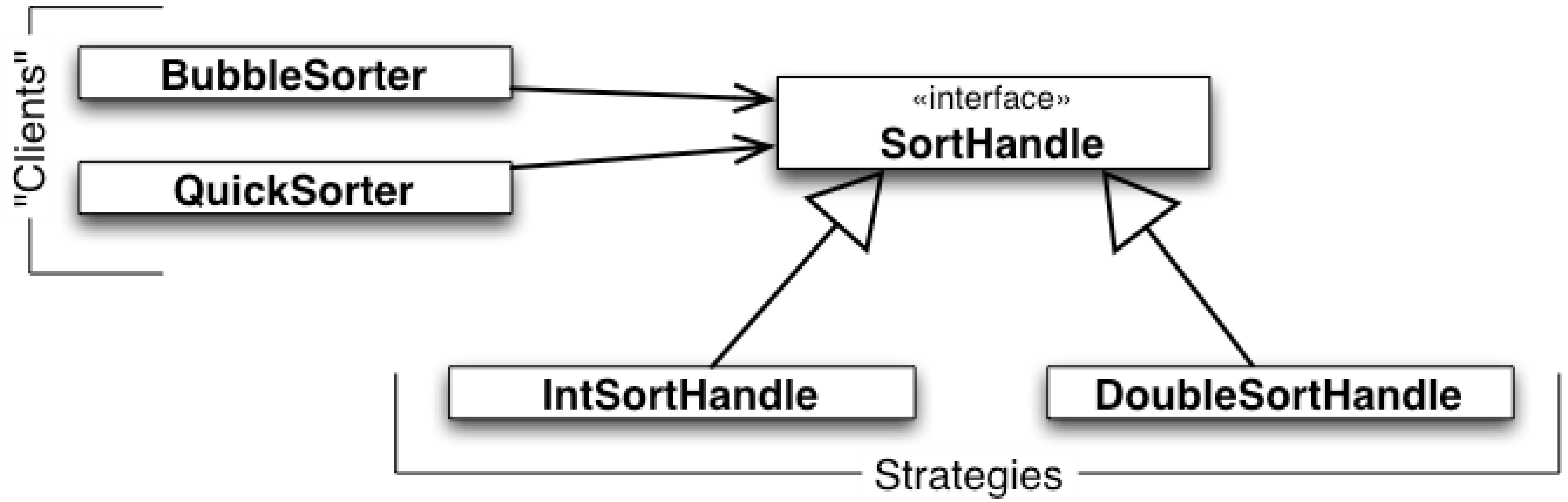
IntBubbleSorter

```
public class IntBubbleSorter extends BubbleSorter{
      private int[] array = null;
      public void sort(int[] theArray) {
            array = theArray;
            length = array.length;
            super.sort();
      }
      protected void swap(int index) {
            int temp = array[index];
            array[index] = array[index+ 1];
            array[index+1] = temp;
      }
      protected boolean outOfOrder(int index) {
            return (array[index] > array[index+ 1]);
      }
}
```

Strategy Pattern



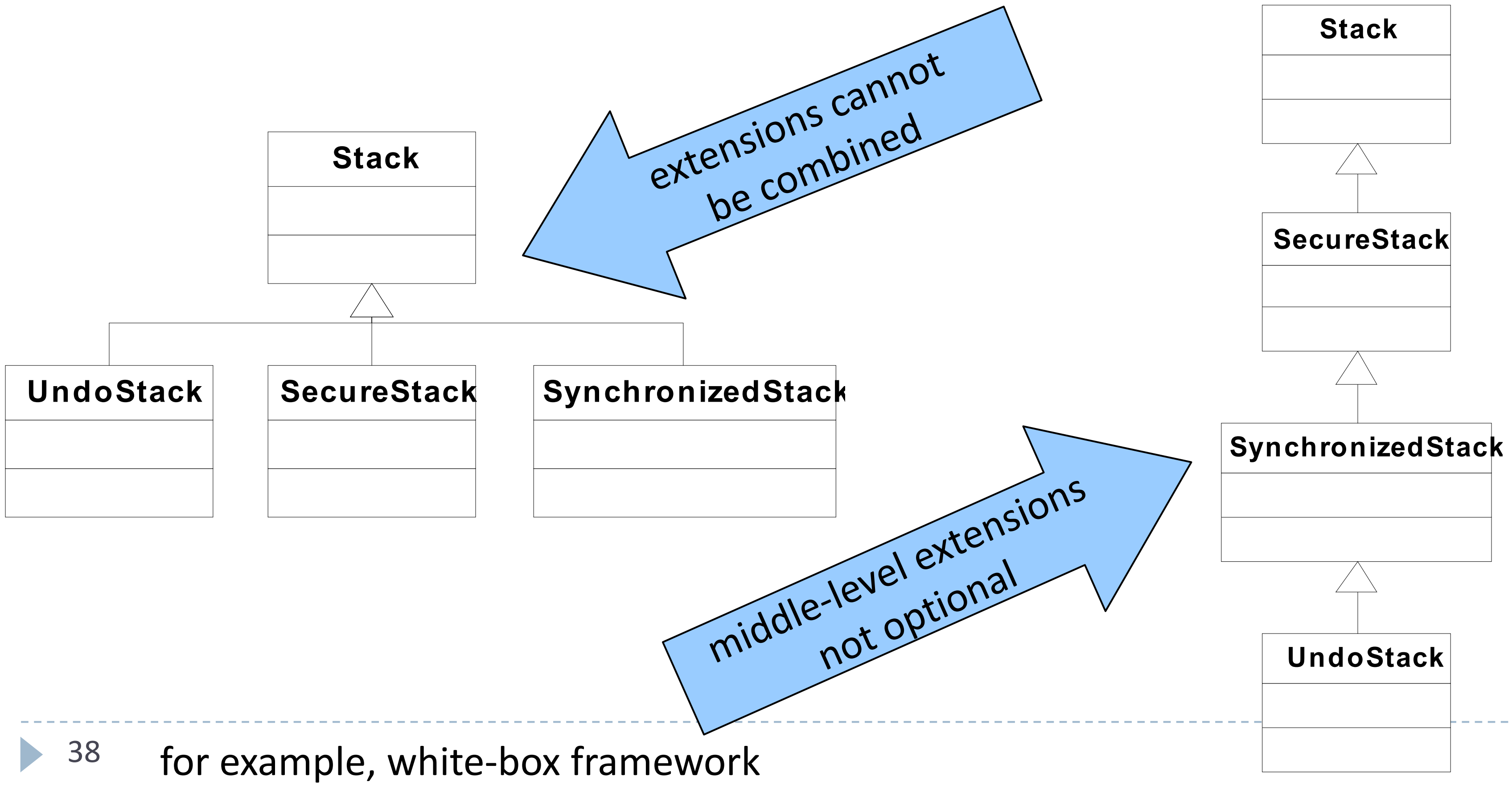
Strategy Pattern: example



Problem:
Inflexible extension mechanisms

Inflexible extension mechanisms

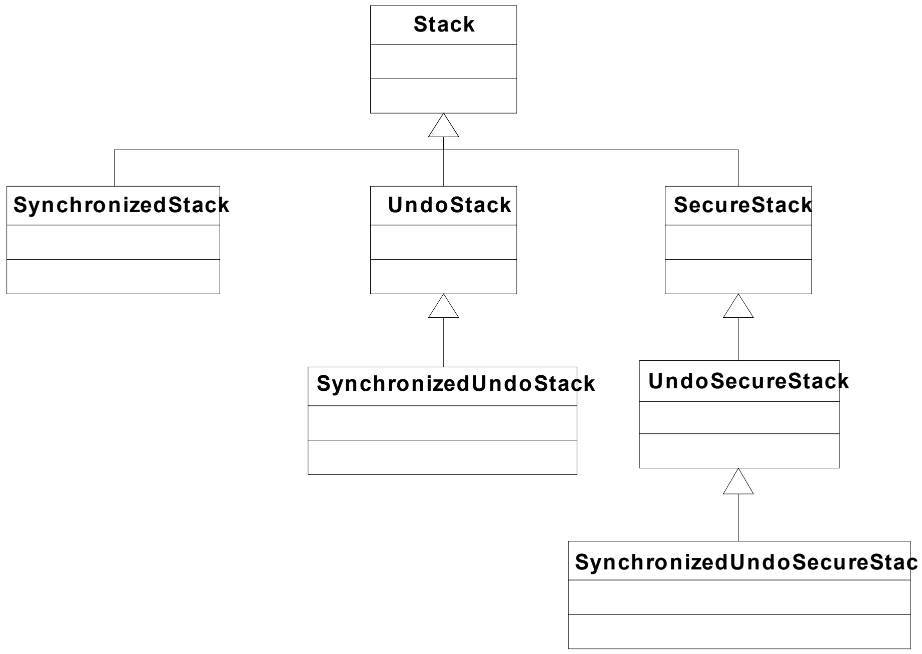
- Subclasses per extension: modular, but inflexible
- No “mix & match”



for example, white-box framework

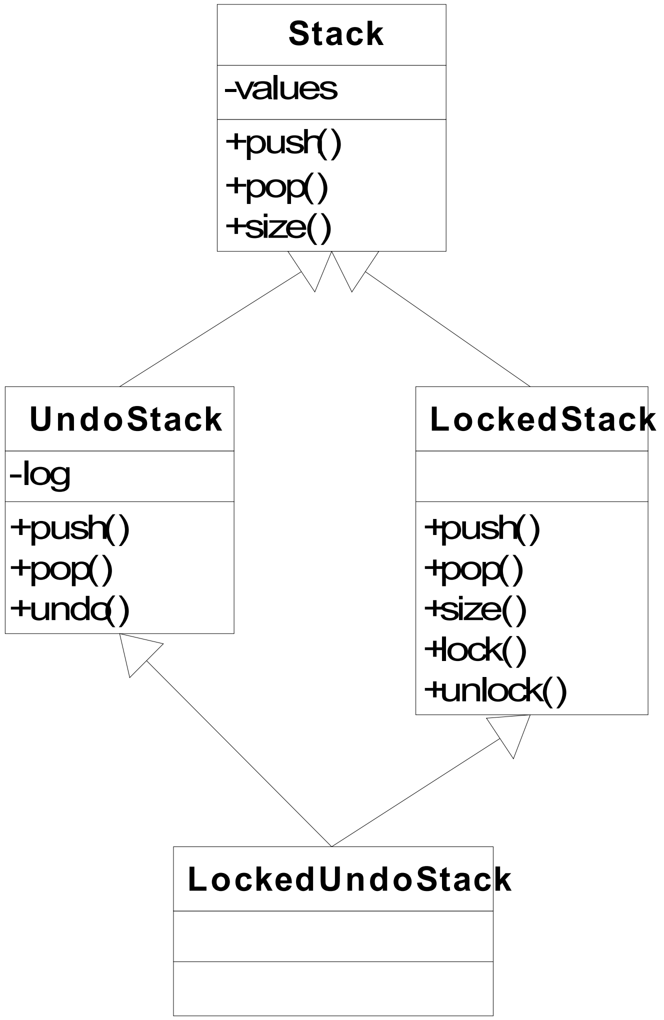
Solution I

- Combined class hierarchies
  - Combinatorial explosion of variants
  - Massive code replication



- Multiple inheritance
  - Combinatorial explosion
  - Due to certain problems (including diamond problem) only available in few languages

Multiple inheritance: diamond problem



What happens?

```
new LockedUndoStack().pop()
```

"Multiple inheritance is good, but there is no good way to do it."
A. SYNDER

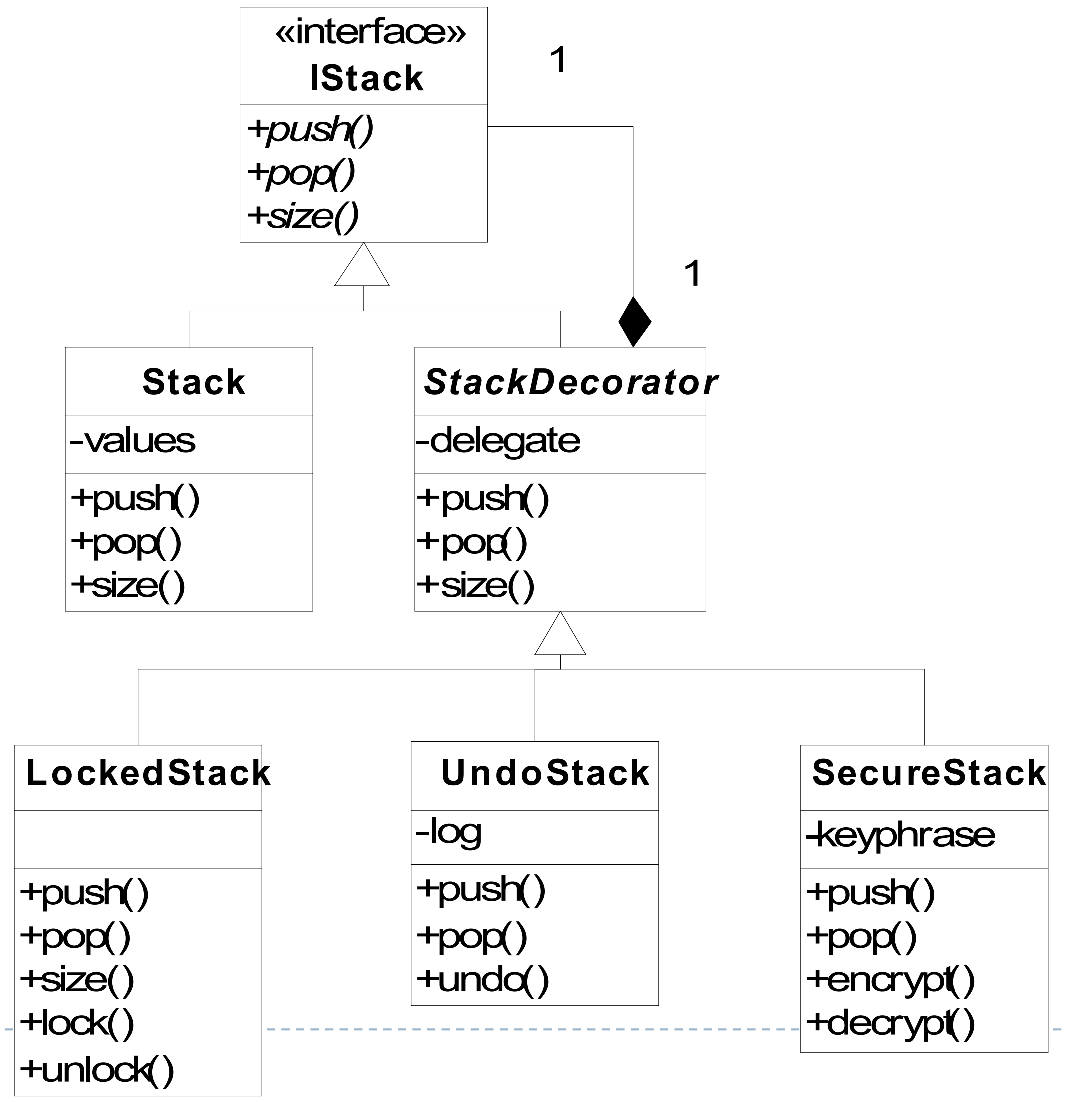
Delegation instead of inheritance

```
class LockedStack implements IStack {
  final IStack _delegate;
  public LockedStack(IStack delegate) {
    this._delegate = delegate;
  }
  private void lock() { /* ... /* }
  private void unlock() { /* ... /* }
  public void push(Object o) {
    lock();
    _delegate.push(o);
    unlock();
  }
  public Object pop() {
    lock();
    Object result = _delegate.pop();
    unlock();
    return result;
  }
  public int size() {
    return _delegate.size();
  }
}
```

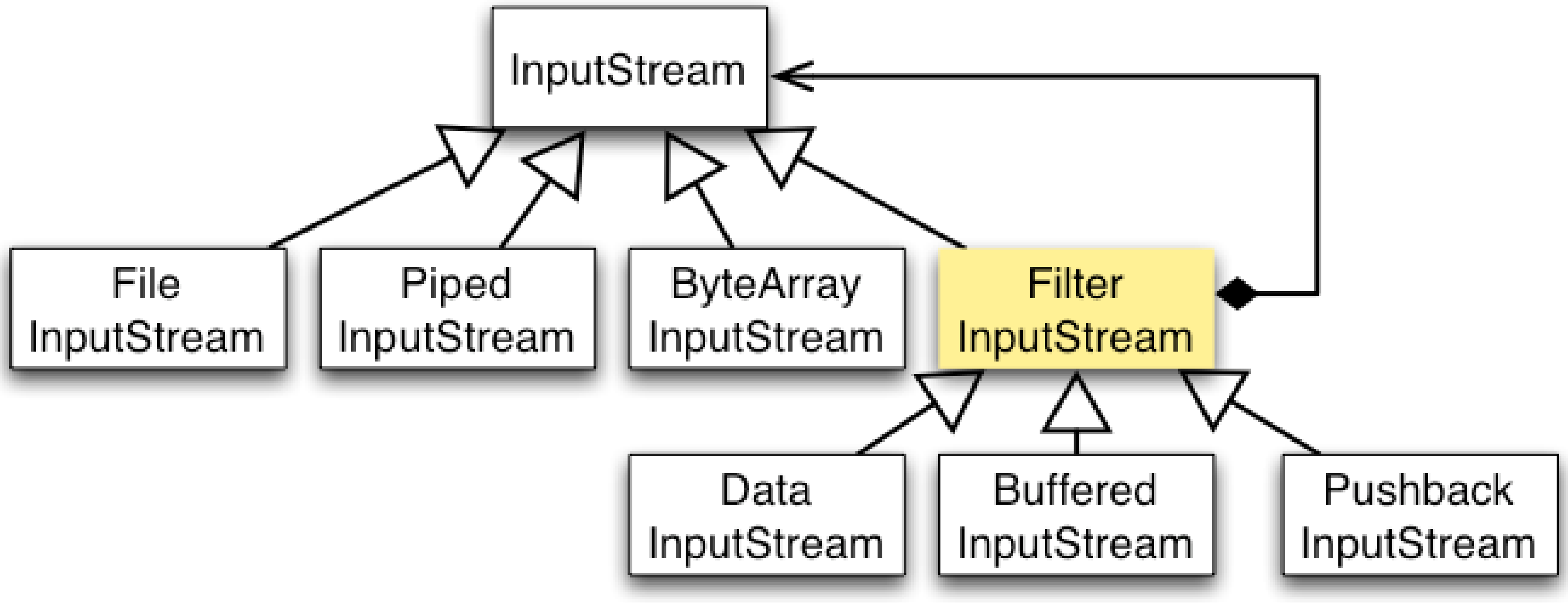
```
class UndoStack implements IStack {
  final IStack _delegate;
  public UndoStack(IStack delegate) {
    this._delegate = delegate;
  }
  public void undo() { /* ... /* }
  public void push(Object o) {
    remember();
    _delegate.push(o);
  }
  public Object pop() {
    remember();
    return _delegate.pop();
  }
  public int size() {
    return _delegate.size();
  }
}
```

```
Main:
IStack stack = new UndoStack(
    new LockedStack(new Stack()));
```

Decorator Pattern



Example: Decorator in java.io



- java.io provides various functions for input and output
  - Programs **operate on stream objects**...
  - **Independent** of data source/target and type of data

Discussion: Delegation instead of inheritance

- Dynamic combination possible
- Extensions have to be independent
- Cannot add methods, only changed existing ones
- No late binding (no virtual methods)
- Many indirections during execution (performance)
- Multiple object instances form an object (object schizophrenia)

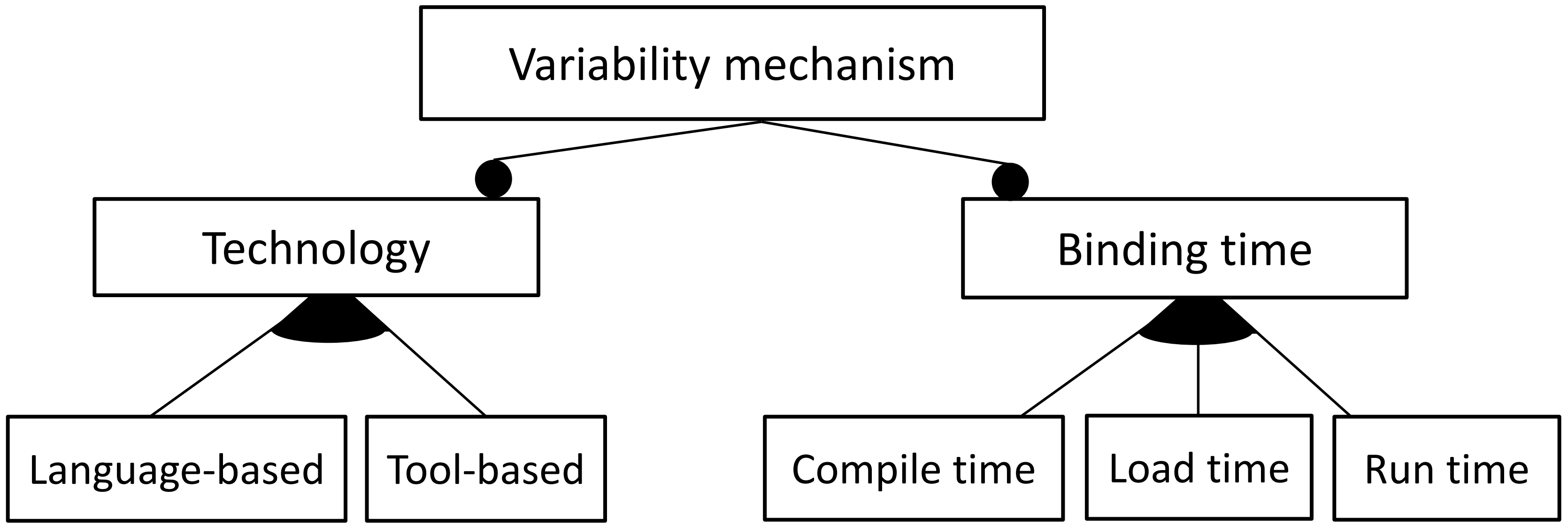
Outlook

- Compile-time variability with generators
- Flexible extension mechanism
- Feature-oriented modularity

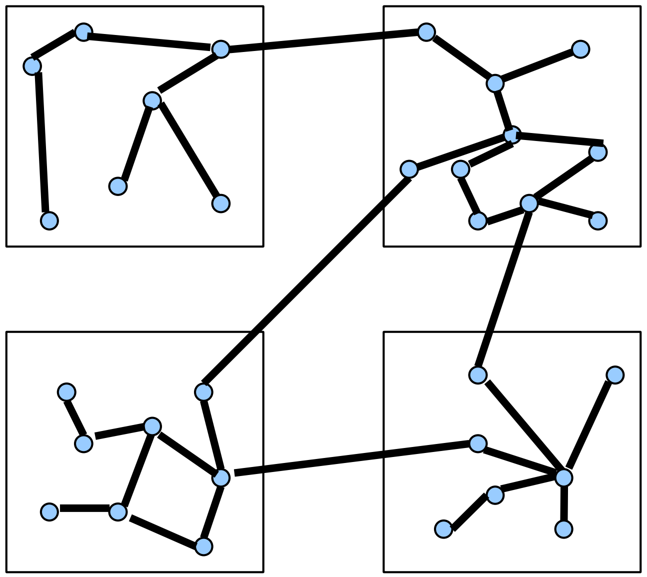
Literatur

- Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.
  [Standard reference for design patterns]
- Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997 – Chapters 3, 4
  [For modularity]

Zoom quiz: design patterns?

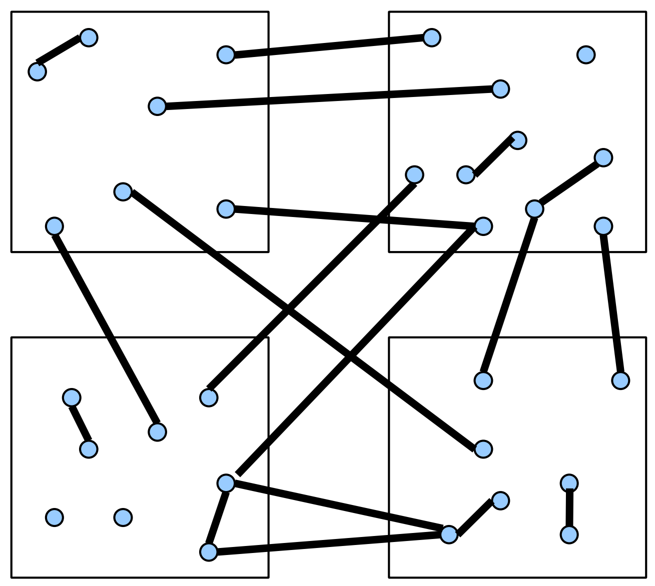


Zoom quiz: Example A



- What is depicted here?
  - (a) strong cohesion + tight coupling
  - (b) strong cohesion + loose coupling
  - (c) weak cohesion + tight coupling
  - (d) weak cohesion + loose coupling

Zoom quiz: Example B



- What is depicted here?
  - (a) strong cohesion + tight coupling
  - (b) strong cohesion + loose coupling
  - (c) weak cohesion + tight coupling
  - (d) weak cohesion + loose coupling