The Use of Overloading in Java Programs

Joseph (Yossi) Gil, Keren Lenz
Technion – Israel Institute of Technology

What is Overloading?
The ability to define more than one method with the same name in a class.

It is good:
- Allows indicating identity of different methods
- Supports concise method names

interface Printer {
    void print(int i);
    void print(char c);
}

It is bad:
- "Different things should have different names"

interface GamblerCowboy {
    void deal(DunGambler g);
    void deal(Card c);
}

Research Objectives
Study the use of overloading in Java programs
- What is the extent of overloading?
- How is overloading typically used?
- Is overloading abused?
- Overloading is considered a pitfall, but do people really fall in it?
- Can we reduce the amount of overloading?

Research Method
- Data Set: the Qualitas Corpus - A collection of 99 open source Java applications
- Evaluation method: Hybrid Evaluation
- Automated phase: Scan corpus for overloading
- Manual phase: classify a sample of these
- Why is a manual phase needed?
- The fragile definition problem - There are patterns which can be easily recognized by humans, but not by computers...

class Point {
    double x, y;
    void setLocation(Point p) {
        x = p.x; y = p.y;
    } // this x and this y are different
    void setLocation(double x, double y) {
        this.x = x; this.y = y; //  this x and this y are the same
    }
}

Reliability questions about the manual phase:
- Manual classification: reproducible?
- Small sample: predictive?
- Reliability measurement method: Human subject research
- Pre-test: for updating and refining the taxonomy
- Categorization test: human raters
- Statistical tests on the results
- Cronbach’s α, Cohen’s κ for internal consistency, 
- Binomial proportion confidence interval
- for sampling error
- Conclusions (from statistical analysis)
- Reproducible? Yes!
- Predictive? Yes!

Overloading Patterns Taxonomy

The good: Intrinsic - methods which relate to their peer both semantically and syntactically.

class Point{
    double x, y;
    void setLocation(Point p)
        setLocation(p.x, p.y);
    void setLocation(double x, double y){
        this.x = x; this.y = y;
    }
}

The bad: Accidental - No relationship between peers could be identified.

class Shape{
    public boolean isSame(Shape o) {...}
}
class Circle extends Shape{
    public boolean isSame(Circle c) {...}
}

The ugly: Dummy-Argument - An argument is used to distinguish between peers.

class Point{
    double x, y;
    Point(double x, double y){
        this.x = x; this.y = y;
    }
    Point(double x, double y, boolean polar) {
        x = cos(90)*r; y = sin(90)*r;
    }
}

Intrinsic Overloading Sub-taxonomy

Intra-class - A method invokes a method with a different name but some arguments of the same class

class cite {
    void addElement(String hashcode,Element elem)
        addElementToRegistry(hashcode,elem);
    }
    void addElement(Element elem)
        addElementToRegistry(elem);
}

Default Arguments - overloading is used as a substitute to default arguments mechanism

class Point{
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
}

Results

- Overloading is extensively used
- 35% of constructors and 14% of methods are overloaded
- Overloading is not abused:

- The most common pattern: Default Arguments

- We have an implementation of default arguments including call by keyword.

Conclusions

- Overloading is usually not abused
- Yet, even good overloading is not desirable
- Confusing semantics
- Adding defaults arguments mechanism to Java can eliminate 25% of the uses of overloading!

For further information
- Read the paper, published in ECOOP'10
- Contact keren@cs.technion.ac.il