Flyweight

Rethabile Mabaso

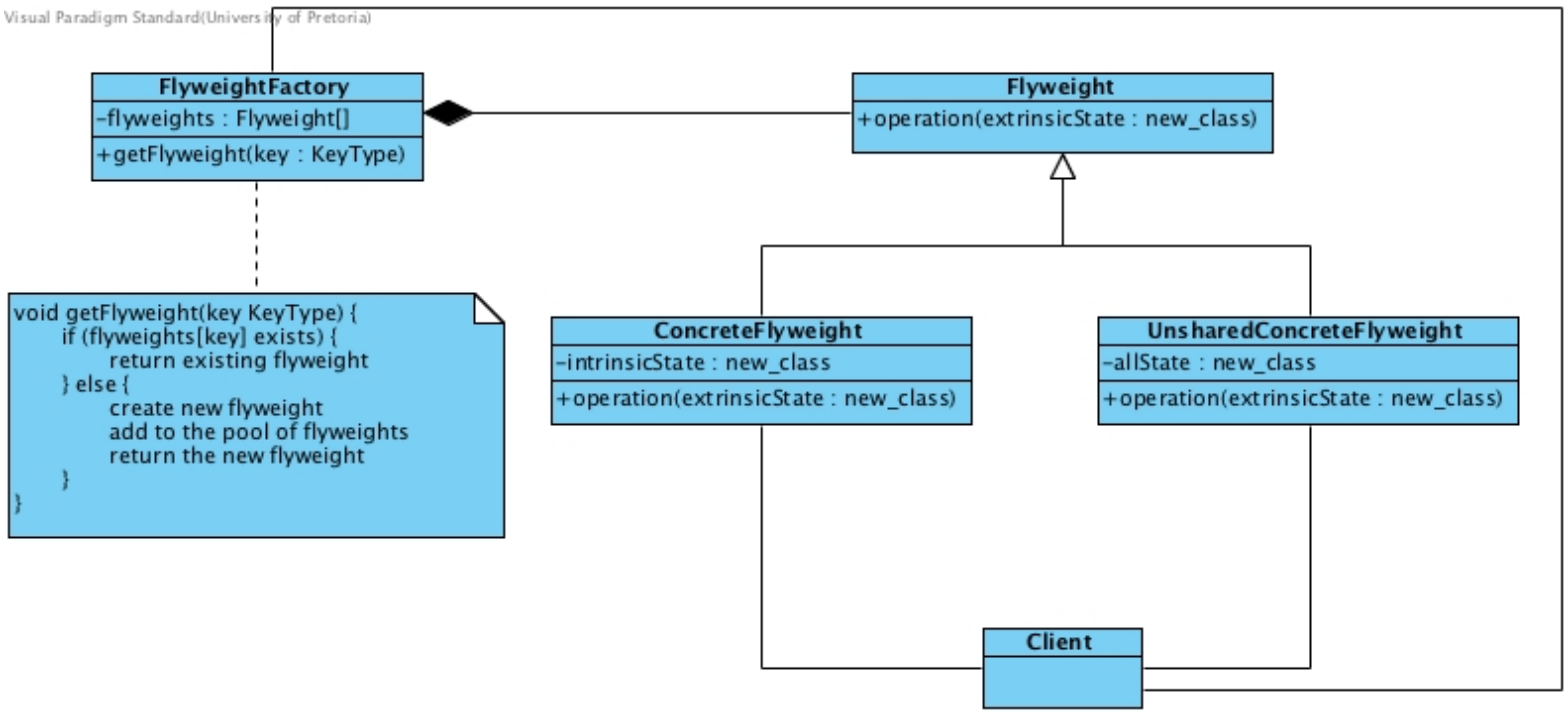
Department of Computer Science
University of Pretoria

15 November 2021

Name and Classification: Flyweight (Object Structural)

Intent: "User sharing to support large numbers of fine-grained objects efficiently." GoF(195)

"User sharing to support large numbers of fine-grained objects efficiently." GoF(195)

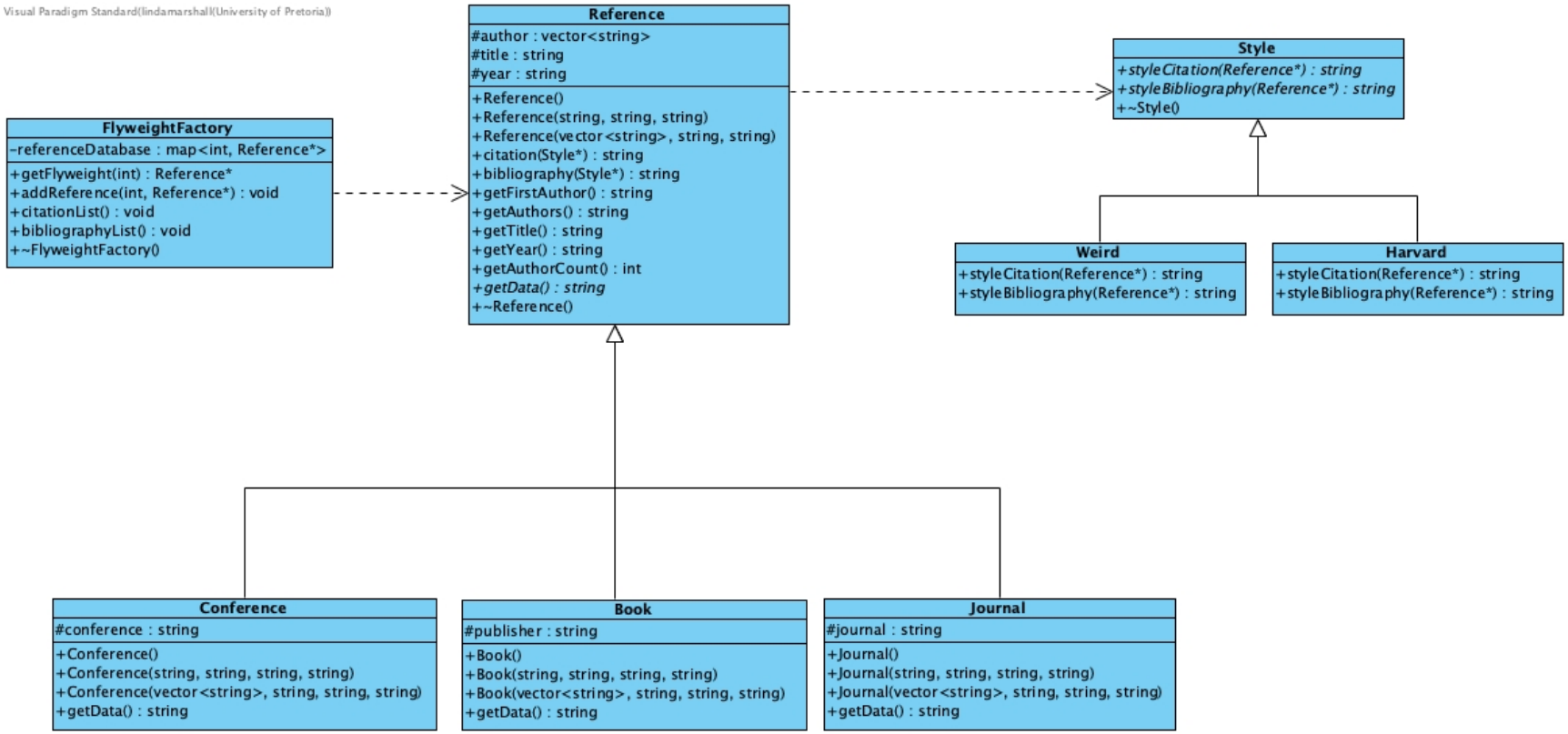

Visual Paradigm Standard(University of Pretoria)
FlyweightFactory
-flyweights : Flyweight[]
+getFlyweight(key : KeyType)
Flyweight
+operation(extrinsicState : new_class)
void getFlyweight(key KeyType) {
if (flyweights[key] exists) {
return existing flyweight
} else {
create new flyweight
add to the pool of flyweights
return the new flyweight
}
}
ConcreteFlyweight
-intrinsicState : new_class
+operation(extrinsicState : new_class)
UnsharedConcreteFlyweight
-allState : new_class
+operation(extrinsicState : new_class)
Client

- **FlyweightFactory** - Creates an instance of a flyweight if it does not exist or supplies an existing one.
- **Flyweight** - Defines the interface through which flyweights are instantiated
- **ConcreteFlyweight** - Implements the interface and adds intrinsic (shareable) state storage.
- **UnsharedConcreteFlyweight** - Not all flyweights need to be shared. Therefore not all need to store intrinsic state and add extrinsic state to represent its entire state. UnsharedConcreteFlyweights may have ConcreteFlyweights as children.

Flyweights have both *intrinsic* and *extrinsic* state.

- *Intrinsic* state refers to the internal state of the flyweight and can be shared as it is independent of the context in which the flyweight is. For example: a flyweight may represent a letter.
- *Extrinsic* state refers to the context in which flyweight is and therefore cannot be shared. For example: flyweights are ordered in terms of the context to form words.

- **Composite** In combination with flyweights, can be used to model directed-acyclic graphs.
- **States** can be implemented as flyweights.
- **Strategies** can also be implemented as flyweights.


Visual Paradigm Standard(lindamarshall(University of Pretoria))
Reference
#author : vector<string>
#title : string
#year : string
+Reference()
+Reference(string, string, string)
+Reference(vector<string>, string, string)
+citation(Style*) : string
+bibliography(Style*) : string
+getFirstAuthor() : string
+getAuthors() : string
+getTitle() : string
+getYear() : string
+getAuthorCount() : int
+getData() : string
+~Reference()
Style
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
+~Style()
FlyweightFactory
-referenceDatabase : map<int, Reference*>
+getFlyweight(int) : Reference*
+addReference(int, Reference*) : void
+citationList() : void
+bibliographyList() : void
+~FlyweightFactory()
Weird
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
Harvard
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
Conference
#conference : string
+Conference()
+Conference(string, string, string, string)
+Conference(vector<string>, string, string, string)
+getData() : string
Book
#publisher : string
+Book()
+Book(string, string, string, string)
+Book(vector<string>, string, string, string)
+getData() : string
Journal
#journal : string
+Journal()
+Journal(string, string, string, string)
+Journal(vector<string>, string, string, string)
+getData() : string

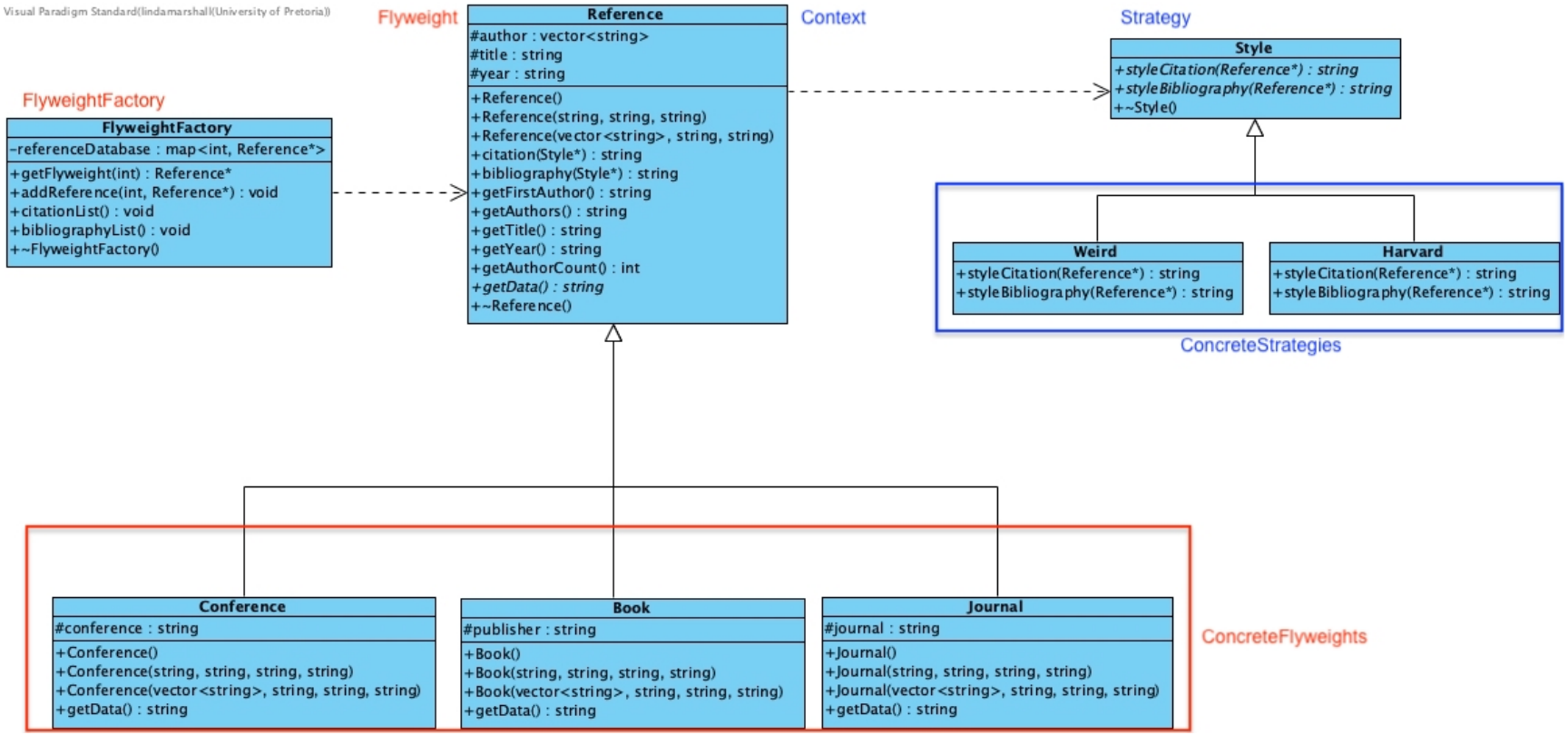
Participants - Flyweight

- FlyweightFactory - FlyweightFactory
- Flyweight - Reference
- ConcreteFlyweight - Journal, Conference, Book

Participants - Strategy

(Provides the extrinsic state)

- Context - Reference
- Strategy - Style
- ConcreteStrategy - Weird, Harvard


Visual Paradigm Standard(lindamarshall(University of Pretoria))
Flyweight
Reference
#author : vector<string>
#title : string
#year : string
+Reference()
+Reference(string, string, string)
+Reference(vector<string>, string, string)
+citation(Style*) : string
+bibliography(Style*) : string
+getFirstAuthor() : string
+getAuthors() : string
+getTitle() : string
+getYear() : string
+getAuthorCount() : int
+getData() : string
+~Reference()
Context
Strategy
Style
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
+~Style()
FlyweightFactory
FlyweightFactory
-referenceDatabase : map<int, Reference*>
+getFlyweight(int) : Reference*
+addReference(int, Reference*) : void
+citationList() : void
+bibliographyList() : void
+~FlyweightFactory()
Weird
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
Harvard
+styleCitation(Reference*) : string
+styleBibliography(Reference*) : string
ConcreteStrategies
Conference
#conference : string
+Conference()
+Conference(string, string, string, string)
+Conference(vector<string>, string, string, string)
+getData() : string
Book
#publisher : string
+Book()
+Book(string, string, string, string)
+Book(vector<string>, string, string, string)
+getData() : string
Journal
#journal : string
+Journal()
+Journal(string, string, string, string)
+Journal(vector<string>, string, string, string)
+getData() : string
ConcreteFlyweights

```
class Style {  // Strategy
  public:
    virtual string styleCitation(Reference*) = 0;
    virtual string styleBibliography(Reference*) = 0;
};

class Harvard : public Style {  // ConcreteStrategy
  public:
    virtual string styleCitation(Reference*);
    virtual string styleBibliography(Reference*);
};

class Weird : public Style { // ConcreteStrategy
  public:
    virtual string styleCitation(Reference*);
    virtual string styleBibliography(Reference*);
};
```

```
string Harvard::styleCitation(Reference* r) {
  string citation("(");

  citation += r->getFirstAuthor();
  if (r->getAuthorCount() > 1) {
    citation += " et al.";
  }
  citation += (","+r->getYear() + ")");
  return citation;
}

string Harvard::styleBibliography(Reference* r) {
  string bibliography;

  bibliography += r->getAuthors()+",";
  bibliography += r->getTitle() + ",";
  bibliography += r->getData() + ",";
  bibliography += r->getYear() + ".";
  return bibliography;
}
```

```
// Acts as the Flyweight and Context for the Strategy
class Reference {
  public:
    Reference();
    Reference(string, string, string);
    Reference(vector<string>, string, string);
    string citation(Style*);
        // gets extrinsic state in and returns in correct form
    string bibliography(Style*);
        // gets extrinsic state in and returns in correct form
    string getFirstAuthor();
    string getAuthors();
    string getTitle();
    string getYear();
    int getAuthorCount();
    virtual string getData() = 0;
  protected:
    vector<string> author; // attributes - intrinsic state
    string         title;
    string         year;
};
```

```
Reference::Reference(){
  title="";
  year = "";
}

Reference::Reference(string a, string t, string y) {
  author.push_back(a);
  title = t;
  year = y;
}

Reference::Reference(vector<string> a, string t, string y) {
  author = a;
  title = t;
  year = y;
}

string Reference::citation(Style* style) {
  return style->styleCitation(this);
}
```

```
string Reference::bibliography(Style* style) {
  return style->styleBibliography(this);
}

string Reference::getFirstAuthor() {
  return author[0];
}

string Reference::getAuthors() {
  vector<string>::iterator it = author.begin();
  string tmp;
  tmp += *it;
  it++;

  for (; it != author.end(); it++) {
    tmp += " and "+(*it);
  }
  return tmp;
}
```

```
string Reference::getTitle() {
  return title;
}

string Reference::getYear() {
  return year;
}

int Reference::getAuthorCount(){
  return author.size();
}
```

```
class Book : public Reference {
  public:
    Book();
    Book(string, string, string, string);
    Book(vector<string>, string, string, string);
    virtual string getData();
  protected:
    string publisher;
};
```

```
Book::Book() :Reference() {
  publisher = "";
}

Book::Book(string a, string t, string p, string y) :
                                    Reference(a,t,y) {
  publisher = p;
}

Book::Book(vector<string> a, string t, string p, string y) :
                                    Reference(a,t,y) {
  publisher = p;
}

string Book::getData(){
  return publisher;
}
```

```
class FlyweightFactory {
  public:
    Reference* getFlyweight(int);
    void addReference(int,Reference*);
    void citationList();
    void bibliographyList();
    virtual ~FlyweightFactory();
  private:
    map<int, Reference*> referenceDatabase;
};
```

```
Reference* FlyweightFactory::getFlyweight(int id){
  map<int,Reference*>::iterator it;

  it = referenceDatabase.find(id);
  if (it != referenceDatabase.end()) {
    return it->second;
  } else {
    cout<<"Must create a flyweight with a dialogue"<<endl;
    return 0;
  }
}

void FlyweightFactory::addReference(int i, Reference* r){
  referenceDatabase.insert(pair<int,Reference*>(i,r));
}
```

```
void FlyweightFactory::citationList() {
}

void FlyweightFactory::bibliographyList(){
}

FlyweightFactory::~FlyweightFactory(){
  // Must release Flyweights (References)
  map<int,Reference*>::iterator it;
  for (it = referenceDatabase.begin();
       it != referenceDatabase.end(); ++it) {
    delete it->second;
  }
}
```