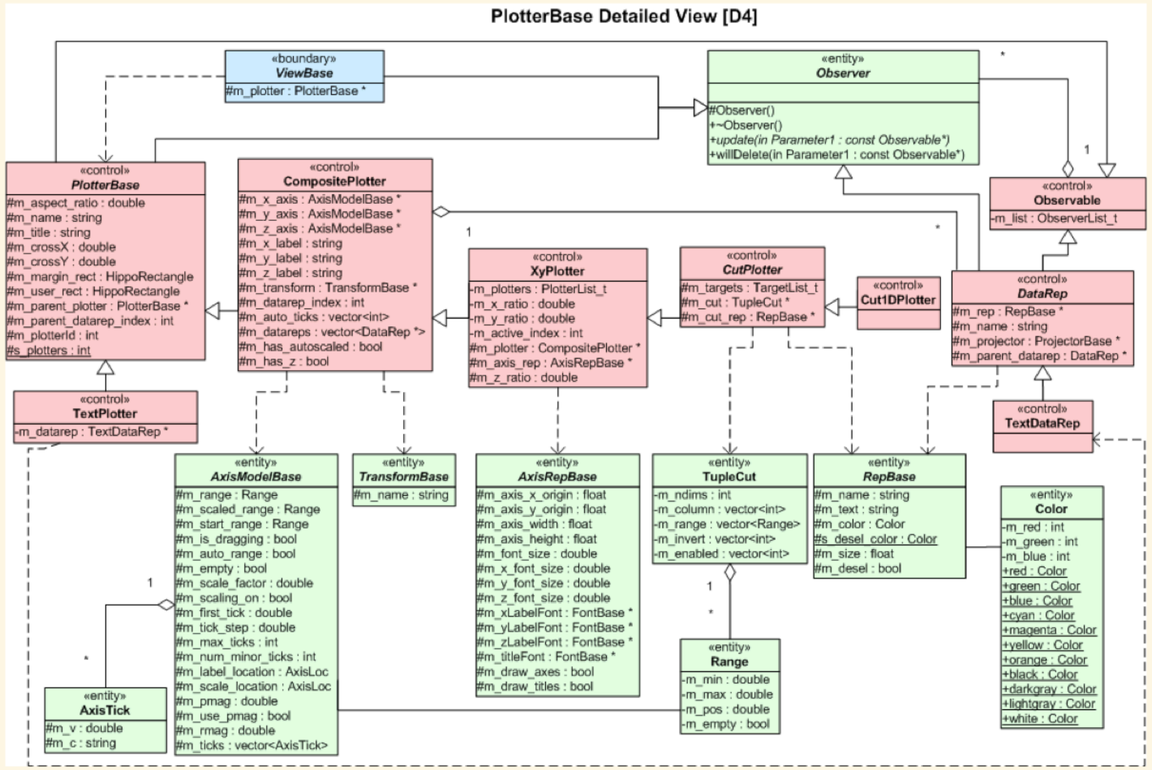
Object-Oriented Programming

Free-Function Stereotypes

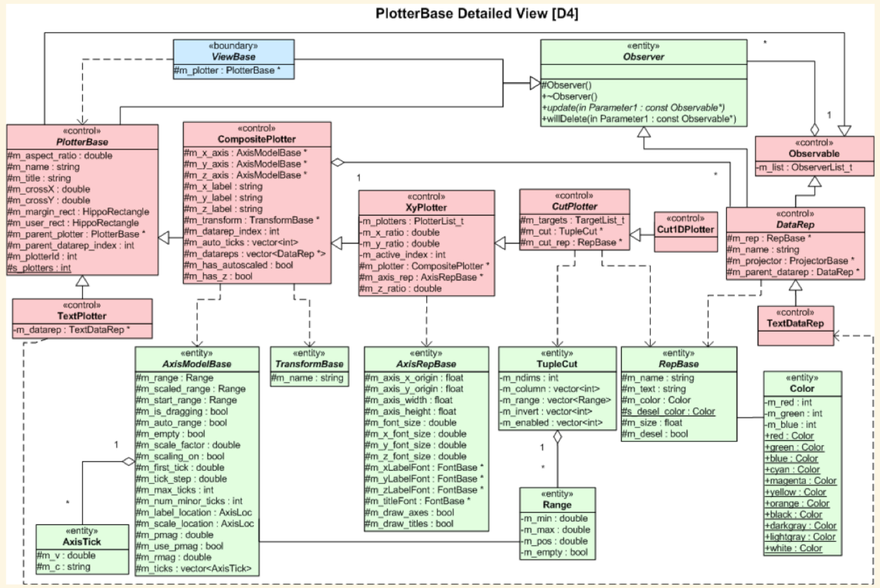
Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

UML Stereotypes



Code Stereotype



- A label we give to code to distinguish it further
- Will look into *method stereotypes* and *class stereotypes* later in the course
- Here, we will look at stereotypes from the perspective of *free functions*

Why Categorize into Stereotypes?

- Functions are a general language feature that is used for many different purposes

- Functions of a particular stereotype share many characteristics and are more similar

- Make sure we have explored the full range of what we can use

- Shorthand for documentation

Free Function Stereotypes

- Accessor::property

- Accessor::predicate

- Mutator::command

Mutator::command

```
void order(int&, int&);

void sort(std::vector<int>& v);

void sort(std::vector<int>::iterator begin,
          std::vector<int>::iterator end);

void remove(std::vector<int>& v,
            int value);
```

- Executes a change based on the IN/OUT parameters

- The function changes the data and parameters are IN/OUT or OUT

- These parameters are *pass by reference* (**not** *pass by const reference*)

- Other parameters may be IN

Accessor::property

```
int average(int, int);

double pay(int hours, double rate);

std::vector<int>::const_iterator search(
    const std::vector<int>& v,
    int value);

std::vector<int>::const_iterator search(
    std::vector<int>::const_iterator begin,
    std::vector<int>::const_iterator end,
    int value);
```

- Returns information derived from the IN parameters

- Parameters are *pass by value* or *pass by const reference*

- When iterators are used, *pass by value* of *const_iterator*

Accessor::predicate

```
bool isValidFilename(std::string_view filename);

bool isStartTag(std::string::const_iterator cursor);

bool contains(const std::vector<double>& v,
              double value);

bool contains(
    std::vector<double>::const_iterator begin,
    std::vector<double>::const_iterator end
    double value);
```

- Returns a Boolean result derived from the IN parameter
- Parameters are all IN
- Must pass all data directly used in the condition
- Useful for even short conditions as it gives them a name
- Often used in refactorings

Accessor::predicate

```
if (content[1] == '!' /* && content[0] == '<' */ &&
    content[2] == '[' && content[3] == 'C' &&
    content[4] == 'D' && content[5] == 'A' &&
    content[6] == 'T' && content[7] == 'A' &&
    content[8] == '[') {
}
```

- Often extracted from code with complex conditional statements
- Anywhere to hide details on how the predicate is determined

Accessor::predicate naming

```
bool empty(/* ... */);

bool contains(/* ... */);

bool isDone(/* ... */);

bool isPDFWord(/* ... */);
```

- Set of standard terms that everyone knows means a predicate

- If not a standard term, then verb form often starts with `is`

- Other verb forms depending on the context

- Do not include the parameter or return types in the name, e.g., no "bool"

- Never use the term "returns" in a function name