

Creating Classes

What we need

- If we want to create object classes ourselves, we need to specify:
 - constructor(s) that can be used to create an instance
 - data members to hold the instance's state
 - the methods that can be used with an instance
 - the behavior of the methods

Basics

- A class that describes an object is basically the same as a code-library class
 - ... except we think of the parts as describing an object, instead of just a collection of functions.
- eg. object for an SFU student:

```
class Student {  
    ...  
}
```

Instance Data

- Class variables/data members that hold the state of the object
 - eg. for a `Student`: name and student number
- Implemented as variables in the **class**.
 - Not in a method, directly inside the `class {...}`
 - ... because it's data associated with the object, not with the implementation of a method

Defining Data Members

- Defined just like other variables, but are not in a method:

```
class Student {  
    private String firstName, lastName;  
    private long studentNumber;  
    ...  
}
```
- These can be used from **any** of the methods in the class.

Encapsulation

- The data members were defined as `private`.
 - Data members should always be `private`.
 - Cannot be used/changed by any code **outside** of the class.
 - eg.

```
Student s = new Student();  
s.firstName = "Rudiger"; // error
```
- All access through getter and setter methods.

Why?

- Why declare all data members as private?
- Since variables can only be changed by the class, the class author can control access.
 - Methods can be designed so the data is always in a useful state.
 - eg. `studentNumber` must be 9 digits, starts with ...
- Other methods can assume meaningful data in those variables.

Why?

- When debugging the class, we don't have to worry about outside code modifying variables.
 - Makes debugging **much** easier.
 - Like local variables in a function/method, but used for the whole instance.
- The object design can then assume that only internal code has changed variables.
 - eg. student number will **always** have 9 digits, ...

Methods


- Just like methods in a code-library class
- **Except:** want to make a new copy for each instance so they are **not** static.
- eg.

```
class Student {
    private String firstName, lastName;
    public void setFName(String name) {
        if ( name.length() > 0 )
            firstName = name;
    }
}
```

Methods

- Methods can be used to construct getter and setters.
 - ... or any other operations that are needed.
 - eg.

```
class Student { ... // declare variables
    public void setFName(String name) {...}
    public String getFName() {...}
    public void addMark(...) {...}
}
```


- public and not static: a visible part of each instance.

Constructors

- Special methods that are called when an instance is created
 - Used to initialize the state of the object.
- Named after the class, no return type:
- eg.

```
class Student {
    private long studentNumber;
    public Student(long stunum) {
        studentNumber = stunum;
    }
}
```

Example

- See `Student.java` and `StudentTest.java` on web site.