Object Oriented Design

Object Oriented Programming in Java

EHSDI

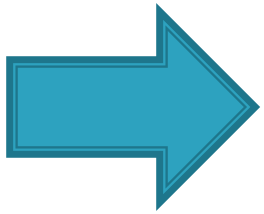
eBuzima

Why OOP?

- OOP features were incorporated into programming languages to help developers produce code that is:
  - Easier to test and so of higher quality
  - Easier to maintain
  - Easier to re-use in future projects
- OOP seeks to achieve these objectives by making code more modularized (i.e. data is stored self-sufficient classes)

Defining the Classes

- When designing your program think about what data should be grouped together, e.g
  - A person's name, address, DOB and other details
  - A coordinate's X and Y values
- Classes are often a model of real world things, i.e. a *User* class models a person…



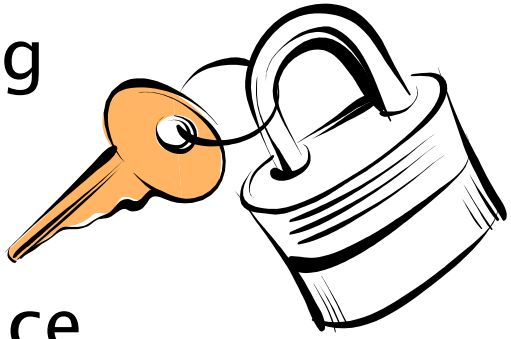
```
class User {
   String name;
   int age;
}
```

Methods and Variables

- We can think of methods as what a class *does* and variables as what a class *knows*
- Methods that are not called by other classes should be *private* or *protected*
- Variables not accessed by other classes should be *private* or *protected*
- Methods should not be too long (50 lines?) as long methods are harder to test and maintain

Encapsulation

- Encapsulation is about concealing the functionality of a class from other classes
- A class should provide an interface of public methods which other classes can call to manipulate that classes data
- This means that the functionality of a class can be changed without breaking the entire program



Getters and Setters

- It's common practice to prevent direct access to instance variables from other classes, and instead provide *getter* and *setter* methods

```
class Shape {
  protected int color;

  public void setColor(int color) {
    this.color = color;
  }
  public int getColor() {
    return color;
  }
}
```

Why Encapsulation?

- We can modify a class without changing how other classes interact with it
- We can make a field read-only by only providing a getter
- We can protect our class from invalid values by checking values in the setter, e.g.

```
public void setColor(int color) {
  if (color >= 0)
    this.color = color;
}
```

Inheritance

- Inheritance enables us to share functionality between different classes, and thus avoid duplication of code
- We should look for generalizations that can be made about our classes, e.g. if we have a *Patient* class and *Doctor* class, and they both have names, DOBs etc, then we could extract that data to a more general superclass called *User*

Naming Conventions

- Class names should start with uppercase and use Camel case
- Methods and variables should start with lowercase and use Camel case
- You should use meaningful names but not too long…

```
int n;                          // Short but meaningless
int numberOfUsersInThisProgram;     // Bit too long
int numUsers;                               // Good!
```

Packages

- Packages in Java are like namespaces in other languages. There are used to:
  - Group related classes together – e.g. all the classes in a particular project
  - Differentiate between classes from different sources – e.g. if you create a class which has the same name as another class in a library you are using

Package Naming

- You can call your packages whatever you like, but if your code is going to be made publically available, you'll want to choose package names that are unique
- Sun recommends that you combine your company's TLD, domain name, and package name, e.g.
  - org.openmrs
  - rw.rita.esoko