

Dating site example

The 7 Stages of Programming

1. Requirements

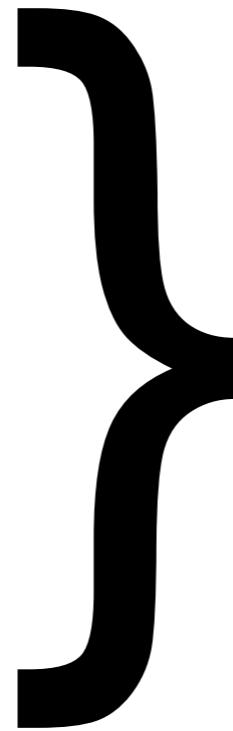
2. Specification

3. Design

4. Implementation

5. Testing

6. Debugging



7. Documentation

Requirements

We want to build a dating site which contains profiles for many users.

Each profile should contain information such as:

- year of birth
- star sign
- Likes dogs / likes cats

Requirements

Users should be able to:

- look-up profiles by name
- filter profiles by matching profile information
- sort profiles by desirable profile information
- block access by other users

Specification: Profiles

Profiles will contain the following information:

- A unique username
- The year of birth
- The star sign
- A 'likes dogs' field
- A 'likes cats' field
- A list of blocked users

Specification:

Look-up

We will provide a fast look-up method which will return the profile of a user with a provided username.

The method will need to know the username of the target and the username of the person asking.

The method will return null if the target does not exist or if the asker is blocked.

Specification: Filtering

There is a method which allows a user to get a list of all unblocked profiles.

They can then filter this list based on criteria:

- filter star sign
- filter by liking cats or dogs

Specification: Sorting

The user will also be able to sort a list of profiles by age.

Design

We will have two main classes:

- A Profile class to record individual profiles and control access to their information.
- A DatingSite class to collect all the Profiles and provide search and filter methods.

Profile fields

```
String myUsername;
```

```
int myBirthYear;
```

```
int myStarSign; // from 1 to 12
```

```
boolean myLikesCats;
```

```
boolean myLikesDogs;
```

```
ArrayList<String> myBlockedUsers;
```

Profile accessor methods

```
String getUsername()
```

```
int getAge()
```

```
int getStarSign()
```

```
boolean likesCats()
```

```
boolean likesDogs()
```

```
boolean isBlocked(String user)
```

Profile mutator methods

```
void setLikesCats (boolean likes)
```

```
void setLikesDogs (boolean likes)
```

```
void blockUser (String username)
```

```
void unblockUser (  
                    String username)
```

DatingSite fields

A map from usernames to their corresponding Profiles:

```
HashMap<String, Profile>  
myProfiles;
```

DatingSite accessor methods

Get a users profile if the requestor is not blocked:

```
Profile getProfile(  
    String requester,  
    String username)
```

DatingSite accessor methods

Get all profiles that are unblocked:

```
ArrayList<Profile>  
getUnblockedProfiles(  
    String requester)
```

Dating Site filter methods

Take a list and filter out the elements that don't match the give criteria. Return the filtered list.

```
ArrayList<Profile> filterLikesDogs (  
    ArrayList<Profile> list,  
    boolean likes)
```

```
ArrayList<Profile> filterStarSign (  
    ArrayList<Profile> list,  
    int sign)
```

DatingSite sortBy method

Take a list and sort it in increasing order of age.
Return the sorted list.

```
ArrayList<Profile> sortByAge(  
    ArrayList<Profile> list)
```

Style: Magic numbers

We often use **numbers** to represent other concepts in our code (eg: star sign).

It is more convenient to use a number than a string, but the numbers on their own have little meaning and thus make code hard to read:

```
mySign = 5;
```

```
// BAD: What sign is this?
```

Style: Constants

To make our code more readable, we use **constants** to give names to these magic numbers:

```
final static int GEMINI = 5;
```

```
final static int CANCER = 6;
```

```
final static int LEO = 7;
```

Style: Constants

To make code more readable, we use
constants and give names to these magic
numbers:

it cannot
change
(read-only)

```
final static int GEMINI = 5;
```

```
final static int CANCER = 6;
```

```
final static int LEO = 7;
```

Style: Constants

To make constants more readable, we use magic numbers:
con

it cannot change (read-only)

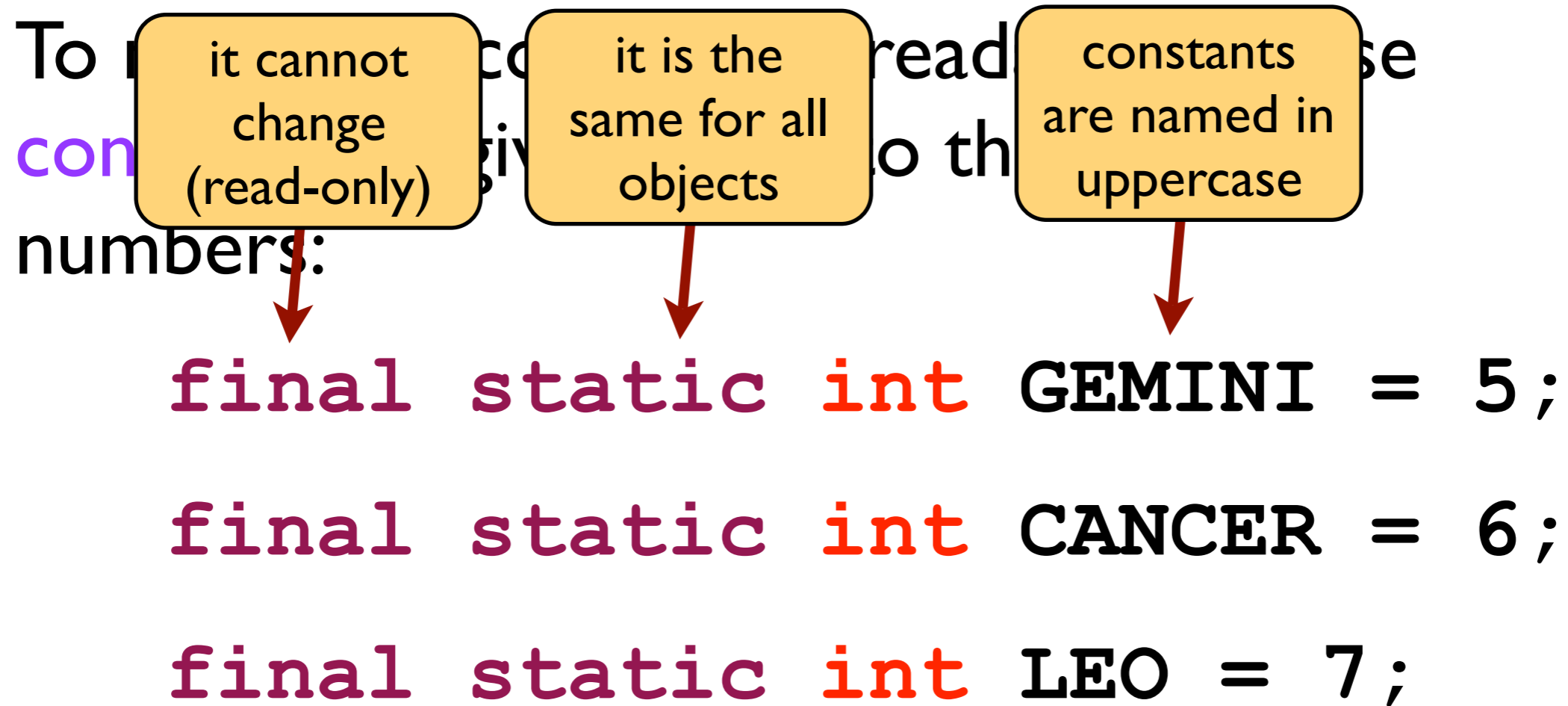
it is the same for all objects

`final static int GEMINI = 5;`

`final static int CANCER = 6;`

`final static int LEO = 7;`

Style: Constants



Constants are final

Constants can be used in expressions like any other variable, but their values **cannot** be changed:

```
mySign = GEMINI;
```

```
// mySign is now 5
```

```
GEMINI = 7; // ERROR
```

Constants are static

Constants belong to the **class** not the a particular object in the class (unlike fields).

Since constants are read-only, we can make them **public** so other classes can use them.

To access a constant on another class we use the syntax:

```
int sign = Profile.GEMINI;
```

Constants are static

Constants belong to the **class** not the a particular object in the class (unlike fields).

Since constants are read-only, we can make them **public** so other classes can use them.

To access a constant er class we use the syntax:

```
int sign = Profile.GEMINI;
```

Constants are static

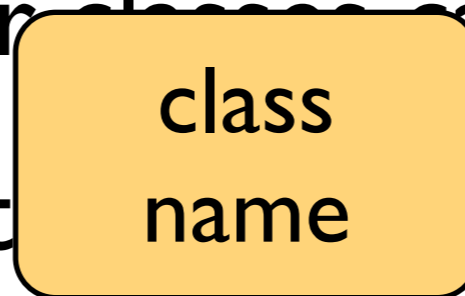
Constants belong to the **class** not the a particular object in the class (unlike fields).

Since constants are read-only, we can make them **public** so other classes can use them.

To access a constant in a class we use the syntax:

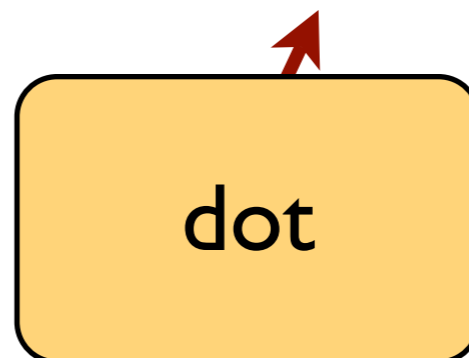
```
int sign = Profile.GEMINI;
```

class
name



A yellow rounded rectangle containing the text 'class name'. A red arrow points downwards from this box to the word 'Profile' in the code example below.

dot



A yellow rounded rectangle containing the text 'dot'. A red arrow points upwards from this box to the dot character '.' in the code example above.

Constants are static

Constants belong to the **class** not the a particular object in the class (unlike fields).

Since constants are read-only, we can make them **public** so other classes can use them.

To access a constant in another class the syntax:

```
int sign = Profile.GEMINI;
```

