App Development & Modeling

BSc in Applied Computing

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

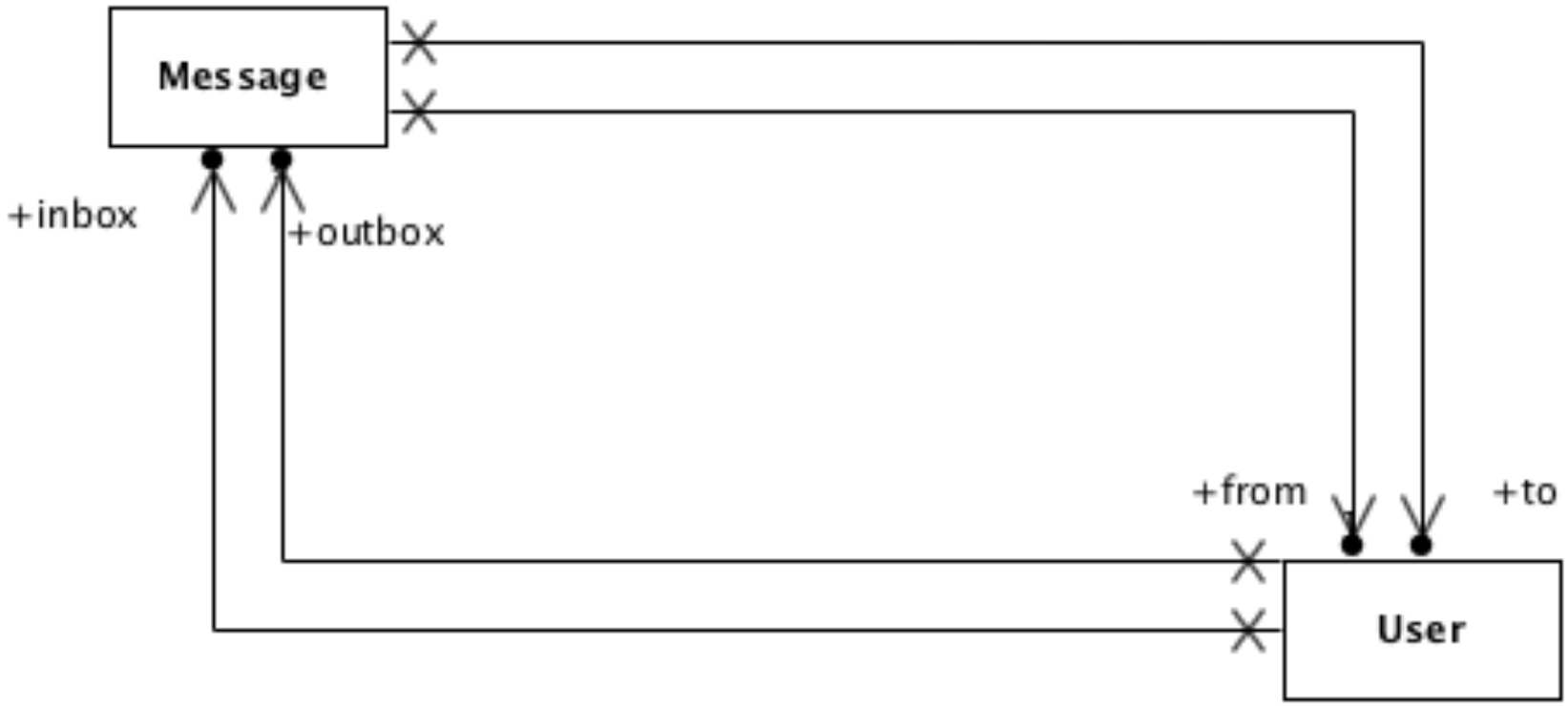
Spacebook Models

User & Messages

```
@Entity
public class User extends Model
{
  public String firstName;
  public String lastName;
  public String email;
  public String password;
  public int    age;
  public String nationality;

  @OneToMany(mappedBy = "to")
  public List<Message> inbox = new ArrayList<Message>();

  @OneToMany(mappedBy = "from")
  public List<Message> outbox = new ArrayList<Message>();

  public User(String firstName, String lastName, String email,
              String password, int age, String nationality)
  {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.password = password;
    this.age = age;
    this.nationality = nationality;
  }
```

```
public class Message extends Model
{
  public String messageText;

  @ManyToOne
  public User from;

  @ManyToOne
  public User to;

  public Message(User from, User to,
                 String messageText)
  {
    this.from = from;
    this.to = to;
    this.messageText = messageText;
  }
}
```

Model

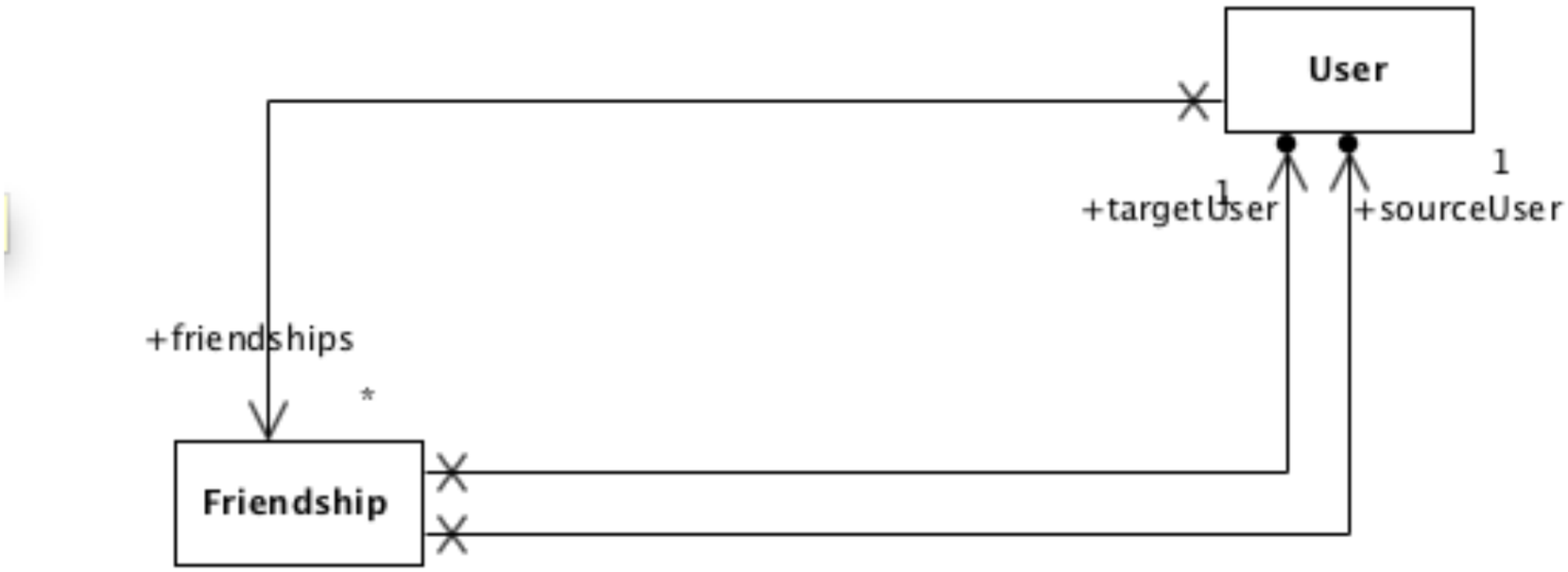


User & Friends

```java
@Entity
public class User extends Model
{
  public String firstName;
  public String lastName;
  public String email;
  public String password;
  public int    age;
  public String nationality;

  @OneToMany(mappedBy = "sourceUser")
  public List<Friendship> friendships = new ArrayList<Friendship>();

  public User(String firstName, String lastName, String email,
              String password, int age, String nationality)
  {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.password = password;
    this.age = age;
    this.nationality = nationality;
  }
```

```java
@Entity
public class Friendship extends Model
{
  @ManyToOne()
  public User sourceUser;

  @ManyToOne()
  public User targetUser;

  public Friendship(User source, User target)
  {
    sourceUser = source;
    targetUser = target;
  }
}
```

Model



Model for Blog

- If a user has a blog, we interpret it to mean that they have posts.
- I.e. A blog consists of one or more ’Posts’
- Each Post consists of:
  - Title
  - Content

Class Post

- A Post is a ‘Model’ class
  - This means it will be ‘persisted’ to the database (like User and Message)
- It consists of:
  - title - string
  - content - string (large object - @Lob)

```
public class Post extends Model
{
  public String title;

  @Lob
  public String content;

  public Post(String title, String content)
  {
    this.title = title;
    this.content = content;
  }
}
```

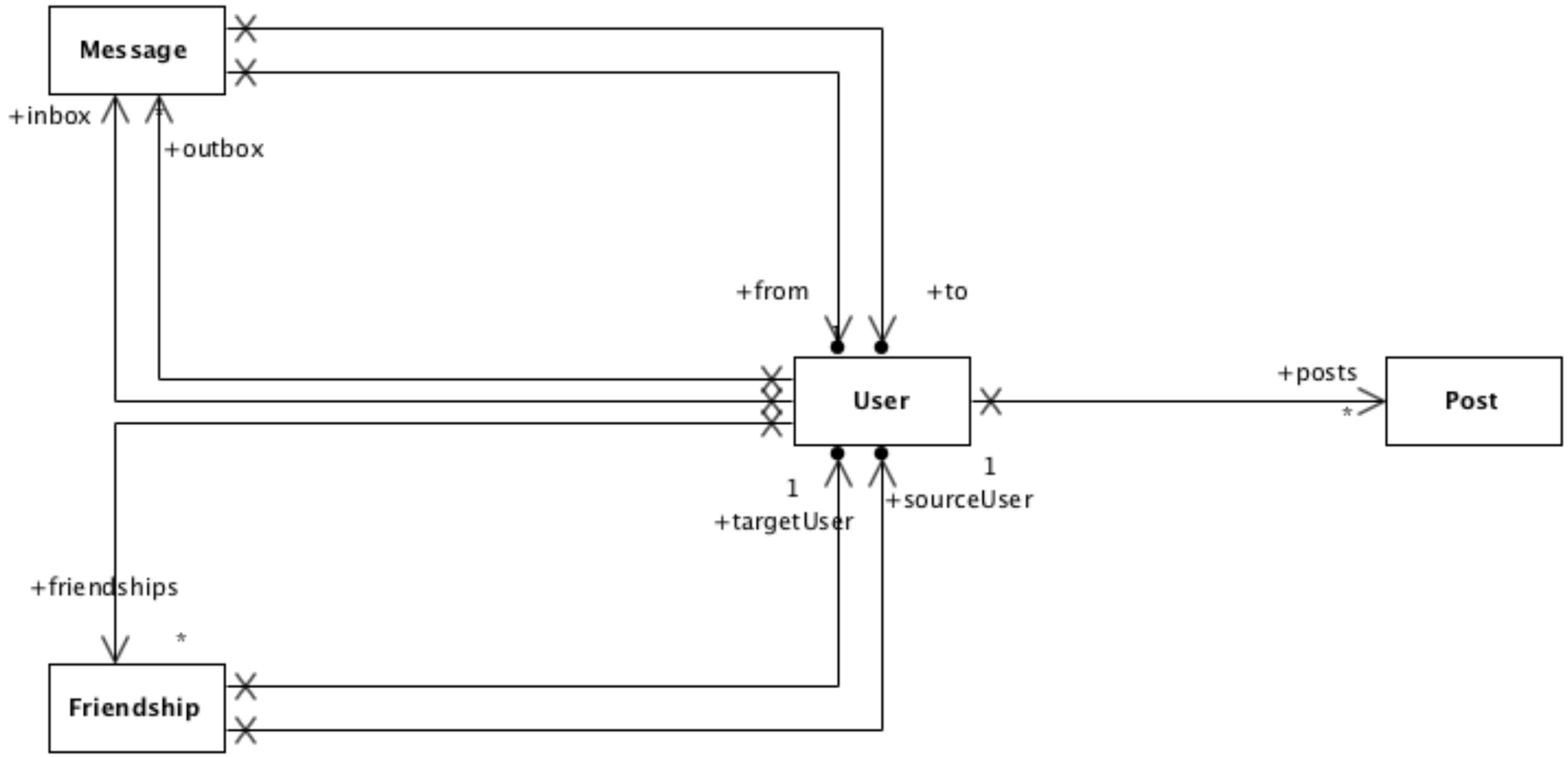
User & Posts

```
@Entity
public class User extends Model
{
  public String firstName;
  public String lastName;
  public String email;
  public String password;
  public int    age;
  public String nationality;

  @OneToMany
  public List<Post> posts = new ArrayList<Post>();

  public User(String firstName, String lastName, String email,
              String password, int age, String nationality)
  {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.password = password;
    this.age = age;
    this.nationality = nationality;
  }
```

```
@Entity
public class Post extends Model
{
  public String title;
  @Lob
  public String content;

  public Post(String title, String content)
  {
    this.title = title;
    this.content = content;
  }

  public String toString()
  {
    return title;
  }
}
```

Model

Complete Model



Write a Test to See if this Works!

```
@Test
public void testCreatePost()
{
  User bob = new User("bob", "jones", 20, "irish", "bob@jones.com", "secret");
  bob.save();

  Post aPost = new Post ("Post Title", "This is the post content");
  bob.addPost(aPost);
  bob.save();

  User user = User.findByEmail("bob@jones.com");
  List<Post> posts = user.posts;
  assertEquals(1, posts.size());
}
```

- Create a User
- Create a Post
- Add the Post to the User and save the user

- See if this user and the post can be ‘read back’

Test Driven Development

- Having created the Post class and made the corresponding changes in User
    - ==> Write unit test **before** proceeding to UI development
- This significantly enhances our chances of getting the Blog feature implemented correctly.
- Otherwise, as we develop the UI, and we encounter problems, we are unsure if these problems are model, controller or view related…

BlogTest - Fixtures

- Whenever we run the tests:
  - Delete all models from database
- Before/after each test
  - run setup() to create some test users + posts
  - ..and remove these after the test has run

```
public class BlogTest extends UnitTest
{
  private User bob;
  private Post post1, post2;

  @BeforeClass
  public static void loadDB()
  {
    Fixtures.deleteAllModels();
  }

  @Before
  public void setup()
  {
    bob   = new User("bob", "jones", "bob@jones.com", "secret", 20, "irish");
    post1 = new Post("Post Title 1", "This is the first post content");
    post2 = new Post("Post Title 2", "This is the second post content");
    bob.save();
    post1.save();
    post2.save();
  }

  @After
  public void teardown()
  {
    bob.delete();
    post1.delete();
    post2.delete();
  }

 //…
}
```

The Tests - testCreatePost

- Assuming the fixtures in place
  - add a new post to bob
  - Look up bob in db
  - see if he has the post we just added

```
@Test
public void testCreatePost()
{
  bob.posts.add(post1);
  bob.save();

  User user = User.findByEmail("bob@jones.com");
  List<Post> posts = user.posts;
  assertEquals(1, posts.size());
  Post post = posts.get(0);
  assertEquals(post.title, "Post Title 1");
  assertEquals(post.content, "This is the first post content");
}
```

The Tests - testCreateMultiplePosts

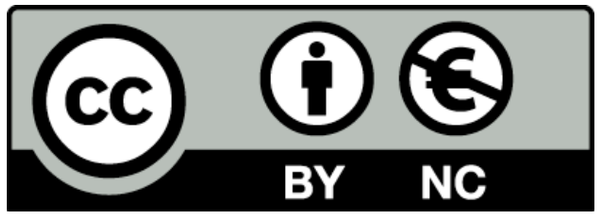
- Add 2 posts
- See if they are there when we look for bob

```
@Test
public void testCreateMultiplePosts()
{
  bob.posts.add(post1);
  bob.posts.add(post2);
  bob.save();

  User user = User.findByEmail("bob@jones.com");
  List<Post> posts = user.posts;
  assertEquals(2, posts.size());
  Post posta = posts.get(0);
  assertEquals(posta.title, "Post Title 1");
  assertEquals(posta.content, "This is the first post content");

  Post postb = posts.get(1);
  assertEquals(postb.title, "Post Title 2");
  assertEquals(postb.content, "This is the second post content");
}
```

The Tests -

- Create a new Post Object (not from fixture)
- save it
- add it to bob
- see if he has it
- remove it
- see if it has been removed

```
@Test
public void testDeletePost()
{
  Post post3 = new Post("Post Title 3", "This is the third post content");
  post3.save();
  bob.posts.add(post3);
  bob.save();

  User user = User.findByEmail("bob@jones.com");
  assertEquals(1, user.posts.size());
  Post post = user.posts.get(0);

  user.posts.remove(0);
  user.save();
  post.delete();

  User anotherUser = User.findByEmail("bob@jones.com");
  assertEquals(0, anotherUser.posts.size());
}
```



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit