Translation and Rotation

- Terms used in graphics
  - Translation - movement in the x or y direction
  - Rotation - rotating around a center point

Question: where are the corners of the triangle?

Programming Translation and Rotation

- Instead of moving and rotating the object, move and rotate the axes!
  - Translation
    - Translate the axes up and right.
  - Rotation
    - Rotate the axes clockwise.

Java Class Hierarchy (review)

```java
public class Person {
    private String name;
    public Person (String theName) {...}
    public String getName() {...}
}
```

```java
public class Student extends Person {
    public Student (String theName) {...}
}
```

Wednesday, February 6, 13
Inheritance (review)

public class Person {
    private String name;
    public Person (String theName) {
    }
    public String getName() {
    }
}

public class Student extends Person {
    public Student (String theName) {
    }
}

Student s;
System.out.println (s.getName());

Adding Behavior to a Subclass (review)

public class Student extends Person {
    private double gpa;
    public Student (String theName) {
    }
    public void setGPA (double newGPA) {
    }
}

Student s;
System.out.println(s.getName());
s.setGPA (3.6);

Person p;
System.out.println(p.getName());
p.setGPA (3.6);

Overriding Methods (Review)

public class Person {
    public Person (String theName) {
    }
    public String toString() {
        return name;
    }
    public String getName() {
    }
    private String name;
}

public class Student extends Person {
    private double gpa;
    public Student (String theName) {
    }
    public void setGPA (double newGPA) {
    }
    public String toString() {
        return getName() + " has gpa " + gpa;
    }
}

A subclass can provide its own definition of an inherited method.

Wednesday, February 6, 13
Polymorphism

A person (in real life) can take on many roles:
- A student
- A friend
- A sister
- A customer

The person you are interacting with may not be concerned about your other roles.
For example, the clerk at Big Y will rarely try to change your GPA!

Polymorphism in Java

- An instance of a subclass can be assigned to a variable of a superclass:
  Person p = new Student("Lucy");

- An instance of a superclass CANNOT be assigned to a variable of a subclass:
  Student s = new Person("Linus");

Assignment to Superclass Variable

Student s = new Student("Lucy");
s.setGPA(3.4);
Person p = s;

"Lucy" 3.4

p
s

Lucy is still a student, even when referenced through p! (This is good!)

Wednesday, February 6, 13
Which toString method?

Student s = new Student("Lucy");
s.setGPA(3.4);
Person p = s;
System.out.println(p.toString());

Should we see Lucy's GPA or not?

Which Method is Called?

Suppose we have code:

object.method();

How does Java know where to look for the method?

1. Look at the class of the value stored in the object variable. Use its "method".
2. If it does not define "method", look in its superclass.

Which toString method is called?

Student s = new Student("Lucy");
s.setGPA(3.4);
Person p = s;
System.out.println(p.toString());

What is the type of the value in p?

Use the toString method defined in (or inherited by) Student.
Designing A Class Hierarchy

- What are the classes in your program?
- Are there classes with similar behavior?
- Move similar parts to a superclass.
- Keep differences in individual subclasses.

class Object

- Every class implicitly "extends Object"
- Object defines:
  - public String toString()
  - public boolean equals (Object other)

Assignment

// Ok to assign an instance of a subclass to
// a variable of a superclass.
Object obj = new Rectangle();

// Not ok to assign here since the type of
// obj is not a Rectangle or a subtype of
// Rectangle
Rectangle rect = obj;

How can we get the rectangle out of obj again???
Casting

// Ok to assign an instance of a subclass to
// a variable of a superclass.
Object obj = new Rectangle();

// Tell Java we expect the value to be a
// Rectangle
Rectangle rect = (Rectangle) obj;

What if it doesn't really contain a Rectangle?
Crashes with ClassCastException

instanceof Operator

// Ok to assign an instance of a subclass to
// a variable of a superclass.
Object obj = new Rectangle();

// Check that it really is the type we expect
// before doing the cast
if (obj instanceof Rectangle) {
    Rectangle rect = (Rectangle) obj;
}

Compile-time types vs.
Run-time types

- Eclipse checks compile-time types as you edit code
- Java virtual machine checks run-time types as you execute code
Compile-time Type Checking

Person p = ...;  // Assume this is valid
System.out.println(p.toString()); ✔ Compiles
p.setGPA(3.5);  ✗ Does not compile

- What is the type of the variable declared to be?
- Does that class have a method with the right signature?

Run-time type checking

Person p = new Student("Lucy");
System.out.println(p.toString()); ✔ Runs & displays GPA
Student s = (Student)p; ✔ Runs
s.setGPA(3.5);
Person p2 = new Person("Linus");
Student s2 = (Student)p2; ✗ Compiles but fails
s2.setGPA(2.5);  at runtime

- What is the type of the object being used?
- Execute the method for that object’s type.

Visibility

- Public - visible anywhere the class is visible
- Private - only visible within the class
- Protected - visible in the class and any subclasses

Instance variables should always be private.
**Abstract Class**

- An abstract class is a class that has at least 1 method that is declared but not defined.
- Example: BankAccounts have different fee schedules depending on a number of factors:
  - The type of account
  - The average balance in the last month
  - The minimum balance in the last month
  - The number of transactions made in the last month

```java
public abstract class BankAccount {
    private Customer customer;
    private int balance;
    private double interestRate;

    public BankAccount (Customer c) {...}
    public void deposit(int amount) {...}
    public void withdraw(int amount) {...}
    public int getBalance() {...}
    public abstract void addMonthlyFee();
}
```

**Bank Accounts**

```java
public abstract class BankAccount {
    private Customer customer;
    private int balance;
    private double interestRate;

    public BankAccount (Customer c) {...}
    public void deposit(int amount) {...}
    public void withdraw(int amount) {...}
    public int getBalance() {...}
    public abstract void addMonthlyFee();
}
```

**Abstract Class Consequences**

- Cannot say “new” for an abstract class
  ```java
  BankAccount acct = new BankAccount (cust);
  ```
- Can declare a variable whose type is an abstract class
  ```java
  BankAccount acct = new SavingsAccount(cust);
  ```
Bank Accounts

public abstract class BankAccount {
    protected Customer customer;
    protected int balance;
    protected double interestRate;

globalBankAccount (Customer c) {...}
global void deposit(int amount) {...}
global void withdraw(int amount) {...}
global int getBalance() {...}
global abstract void addMonthlyFee();
}

SavingsAcct acct;
...
acct.balance = 0;

Interfaces vs. Abstract Classes vs. Classes

- interfaces - only contain declarations of methods, no bodies
- classes - complete implementations of all methods
- abstract classes - mix of complete implementations and declarations only
- All 3 can be used as types
- Only classes can be constructed