Designing & Overloading Methods

COMP 110
Summer II 2012

Peter Lincoln
7/12/2012
Outline

- Designing Methods
  - Case Study
  - Testing

- Overloading Methods
Tetris

The relentless building block video puzzle.

FROM RUSSIA WITH FUN!

Official Nintendo Seal of Quality
Get into small groups

- Decide what high-level tasks are required for Tetris gameplay to work

- Assume the graphical display code is taken care of for you
Tetris high-level gameplay tasks

- Choose a random tetromino to give the user
- User-controlled tetromino manipulation
- Game-controlled tetromino manipulation (automatically falling)
- Remove full horizontal lines of blocks
- Increase user score, level, and speed of falling blocks
- Check if game is over
User-controlled tetromino manipulation

- High-level task: manipulate tetromino based on user input
- How can a tetromino be manipulated?
  - Move
  - Rotate
Moving a tetromino

- How?

- Subtasks
  - Move left
  - Move right
  - Move down
Rotating a tetromino

Subtasks

- Rotate left
- Rotate right
Imagine a Tetromino class

```java
public class Tetromino {
    private int x;
    private int y;
    // some other stuff describing this Tetromino's shape

    public void moveLeft() {
        x--;
    }

    public void moveRight() {
        x++;
    }

    public void moveDown() {
        y--;
    }
}
```
Top-down design

- Divide and conquer
- Start with a big problem
- Decompose problem into smaller subtasks
- Decompose big subtasks into even smaller subtasks
- Solve subtasks to solve big problem
Game-controlled tetromino manipulation

How can we implement automatically falling tetrominoes?

What are we trying to do at a high level?

- Every n amount of time, make a tetromino move down one space
- Need a timer
Using the Tetromino class in a game loop

```java
public class TetrisGame {
    private Tetromino userTetr;

    // gameUpdate() is called once per game loop
    public void gameUpdate() {
        // ...do some stuff here
        // check user input, assume userTetr has been properly instantiated
        if (userInput == LEFT) 
            userTetr.moveLeft();
        else if (userInput == RIGHT)
            userTetr.moveRight();
        else if (userInput == DOWN)
            userTetr.moveDown();
        applyAutoFalling(userTetr);

        // do some other stuff here
    }
}
```
public void applyAutoFalling(Tetromino tetr) {
    double timeSinceLastAutoFall =
        // some code to figure out the time since the last fall
    if (timeSinceLastAutoFall > 0.5) {
        tetr.moveDown();
    }
}
What if we see this behavior?

- Imagine that we have run the game
  - A new tetromino appears
  - The user does not provide any input
  - The tetromino does not automatically fall, it simply stays where it is

- What could the problem be?
Let's check `applyAutoFalling`

```java
public void applyAutoFalling(Tetromino tetr) {
    double timeSinceLastAutoFall = // some code to figure out the time since the last fall

    if (timeSinceLastAutoFall > 0.5) {
        tetr.moveDown();
    }
}
```

What if we had this code?

```java
double timeSinceLastAutoFall = 0.0;
```
The problem could be elsewhere

- What if we had this code inside the class Tetromino?

```java
public void moveDown()
{
    y = y;
}
```

- The `moveDown()` method does not do what it is supposed to
Testing

- If a subtask (method) does not work, your solution is incorrect
- Test EVERY method you write
Bottom-up testing

- How do we determine if the error is in applyAutoFalling or moveDown?

- Test each method individually
  - If method A calls method B, fully test method B before testing method A
  - In this case, fully test moveDown before testing applyAutoFalling
Driver programs

- Simple program for only you to test with
  - Run by you, not your user

- Call methods with different inputs
  - Test cases, edge conditions
    - Positive, negative, zero
    - true, false
    - Strings, characters

- MathUtils example in Eclipse
Overloading

- Using the same method name for two or more methods *within the same class*

- We have seen this for constructors already

- Parameter lists must be different
  - public double average(double n1, double n2)
  - public double average(double n1, double n2, double n3)

- Java knows what to use based on the number and types of the arguments
Method signature

- A method’s name and the number and types of its parameters

- signature does NOT include return type

- Cannot have two methods with the same signature in the same class
Gotcha

- Overloading and automatic type conversion

- Imagine we have this constructor defined:

  ```java
  public Pet(double initialWeight)
  ```

- We create a Pet like this:

  ```java
  Pet myPet = new Pet(35);
  ```

- What happens?
Gotcha

- Imagine we have these two constructors defined:
  
  ```java
  public Pet(int initialAge)
  public Pet(double initialWeight)
  ```

- We create a Pet like this:
  
  ```java
  Pet myPet = new Pet(35);
  ```

- What happens?
  - We create a pet with age 35, instead of weight 35.0
Questions?
Logistics

- Next:
  - Open Lab Time
    - Lab 6
    - Program 3

- Tomorrow:
  - Inheritance

- Assignments
  - Program 3 is due Tomorrow
  - Lab 6 is due Monday