Privacy Leaks and Copying Objects

Inspired by material from:
Nelson Padua-Perez, Ben Bederson, Bonnie Dorr, Fawzi Emad,
David Mount, and Jan Plane
Review: private vs. public

- What is the difference between private and public instance variables?

```java
touch class Fraction {
    private int numerator;
    private int denominator;

    public int getNumerator() {
        return numerator;
    }

    public int getDenominator() {
        return denominator;
    }

    public static Fraction neg(Fraction frac) {
        return new Fraction(-frac.getNumerator(), frac.getDenominator());
        // Is the following allowed?
        // return new Fraction(-frac.numerator, frac.denominator);
    }
}

Fraction frac = new Fraction();
frac.numerator = 1;       // Will this give an error?
```
Privacy leaks

Public class GasTank {
    private Fraction fuel;
    public GasTank() {
        fuel = new Fraction(1, 1);
    }
    public Fraction getFuel() {
        return fuel;
    }
    public void setFuel(Fraction f) {
        if (f.asDouble <= 1)
            fuel = f;
    }
}

// What does the following code do?
GasTank tank = new GasTank();
Fraction fuelRead = tank.getFuel();
fuelRead.setNumerator(2);
Privacy leaks

Public class GasTank {
    private Fraction fuel;
    public GasTank() {
        fuel = new Fraction(1, 1);
    }
    public Fraction getFuel() {
        return fuel;
    }
    public void setFuel(Fraction f) {
        if (f.asDouble <= 1)
            fuel = f;
    }
}

// What does the following code do?
GasTank tank = new GasTank();
Fraction fuelRead = tank.getFuel();
fuelRead.setNumerator(2);
Privacy leaks

Public class GasTank {
    private Fraction fuel;
    public GasTank() {
        fuel = new Fraction(1, 1);
    }
    public Fraction getFuel() {
        return fuel;
    }
    public void setFuel(Fraction f) {
        if (f.asDouble <= 1)
            fuel = f;
    }
}

// What does the following code do?
GasTank tank = new GasTank();
Fraction fuelRead = tank.getFuel();
fuelRead.setNumerator(2);
Privacy leaks and fixing them

- A privacy leak occurs when a private instance variable can be modified outside of its class.
- This happens because of aliasing, two references to the same object.
- What if we rewrite `getFuel()`?

  ```java
  public Fraction getFuel() {
    return new Fraction(fuel);
  }
  ```

  - Returns a copy of fuel.
  - Changes to this copy won’t affect the original.
A privacy leak occurs when a private instance variable can be modified outside of its class.

This happens because of aliasing, two references to the same object.

What if we rewrite `getFuel()`?

```java
public Fraction getFuel() {
    return new Fraction(fuel);
}
```

- Returns a copy of `fuel`
- Changes to this copy won’t affect the original
Copying objects

• Three ways to copy objects
  – Reference copy
  – Shallow copy
  – Deep copy

• Let’s start by looking at how to copy an ArrayList of Fractions each way
ArrayList\<Fraction\> f =
   new ArrayList\<Fraction\>();

f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList\<Fraction\> g = f;
ArrayList\<Fraction\> f =
new ArrayList\<Fraction\>();
f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList\<Fraction\> g = f;

- We only copied a reference to an ArrayList
ArrayList<Fraction> f = new ArrayList<Fraction>();
f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList<Fraction> g = new ArrayList<Fraction>();
for (Fraction frac : f) {
    g.add(frac);
}
Shallow copy

ArrayList<Fraction> f =
   new ArrayList<Fraction>();

f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList<Fraction> g =
   new ArrayList<Fraction>();

for (Fraction frac : f) {
   g.add(frac);
}

• We created a new ArrayList, but copied references to each Fraction
Deep copy

```java
ArrayList<Fraction> f =
    new ArrayList<Fraction>();
f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList<Fraction> g =
    new ArrayList<Fraction>();
for (Fraction frac : f) {
    g.add(new Fraction(frac)); // New!
}
```

Stack

Heap

(f)

1
2
3
4
Deep copy

```
ArrayList<Fraction> f =
    new ArrayList<Fraction>();
f.add(new Fraction(1, 2));
f.add(new Fraction(3, 4));

ArrayList<Fraction> g =
    new ArrayList<Fraction>();
for (Fraction frac : f) {
    g.add(new Fraction(frac));  // New!
}
```

- We created a new ArrayList and created new copies of each object in the original ArrayList.
This reference and copy constructors

• The “this” reference represents a reference to the current object being operated on by a non-static method
  – It has automatically been initialized for you
  – It cannot be used in a static method

• A copy constructor is a special type of constructor that takes an instance of the same class as a parameter
  – It uses the data from the parameter object to initialize a new object
  – Example:
    • String s = “example”;
    • String t = new String(s);
public class Fraction {
    private int numerator;
    private int denominator;

    public Fraction(int num, int den) {
        numerator = num;
        denominator = den;
    }

    public Fraction(Fraction f) {
        this(f.numerator, f.denominator);
    }
}

public class GasTank {
    private Fraction fuel;

    public GasTank(Fraction fuel) {
        this.fuel = fuel;
    }

    public GasTank(GasTank tank) {
        this(new Fraction(tank.fuel);
    }

    public Fraction getFuel() {
        return new Fraction(fuel);
    }

    public void setFuel(Fraction f) {
        if (f.asDouble <= 1) {
            fuel = f;
        }
    }
}
Copy comparison

• Which type of copy offers the most protection against aliasing and privacy leaks?
  – Deep copy

• Which type uses the most time and space?
  – Deep copy

• Which type uses the least time and space?
  – Reference copy

• What if the object can’t be modified?
  – These are called immutable objects
Mutable vs. Immutable objects

- A mutable object is changeable (root word: “mutate”)
  - It can have:
    - Setters and other methods that modify instance variables
    - Public instance variables
  - Example: Fraction object

- An immutable object cannot be changed or modified after creation
  - It cannot have:
    - Setters and other methods that modify instance variables
    - Methods that return a reference to a mutable instance variable
    - Public instance variables (instance variables must be private
  - Example: String object
Immutable object example

Public final class ImmutableFraction {
    private final int numerator;
    private final int denominator;

    public Fraction(int num, int den) {
        numerator = num;
        denominator = den;
    }

    public int getNumerator() {
        return numerator;
    }

    public int getDenominator() {
        return denominator;
    }

    public String toString() {
        return numerator + "/" + denominator;
    }
}
Another way to protect GasTank

Public class GasTank {
    private ImmutableFraction fuel; // Now using an ImmutableFraction
    public GasTank() {
        fuel = new Fraction(1, 1);
    }
    public Fraction getFuel() {
        return fuel; // Returns a reference to an object which cannot
                     // be modified
    }
    public void setFuel(Fraction f) {
        if (f.asDouble <= 1)
            fuel = f;
    }
}

// What does the following code do now?
GasTank tank = new GasTank();
Fraction fuelRead = tank.getFuel();
fuelRead.setNumerator(2); // Not possible, ImmutableFraction has no setters!
Another “this” example

Public class PersonWithANameAndAge {
    private String name;
    private int age;

    public PersonWithANameAndAge (String name, int age) {
        this.name = name;
        this.age = age;
    }

    public PersonWithANameAndAge (PersonWithANameAndAge p) {
        this(new String(p.name), p.age);
    }

    public Fraction getName() {
        return new String(name);
    }
}
Code example

- Lets look at some CopyLectureCode.zip
  - Will be available on ELMS