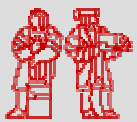
Lecture 8
Classes and Objects Part 2

MIT AITI

June 15th, 2005

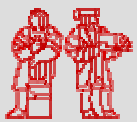
What is an object?

- A building (Strathmore university)
- A desk
- A laptop
- A car
- Data packets through the internet



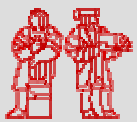
What is an object?

- Objects have two parts:
  - State: Properties of an object.
  - Behavior: Things the object can do.
- Car Example:
  - State: Color, engine size, automatic
  - Behavior: Brake, accelerate, shift gear
- Person Example:
  - State: Height, weight, gender, age
  - Behavior: Eat, sleep, exercise, study



Why use objects?

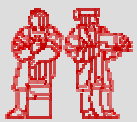
- Modularity: Once we define an object, we can reuse it for other applications.
- Information Hiding: Programmers don't need to know exactly how the object works. Just the interface.
- Example:
  - Different cars can use the same parts.
  - You don't need to know how an engine works in order to drive a car.



Classes

- A class is a template or pattern from which objects are created

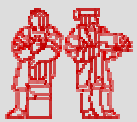
- A class contains
  - Data members (Properties/Characteristics of the objects/class)
  - Methods (Determines the behavior of the objects created from the class)
  - Constructor (Special Method)



Anatomy of a class

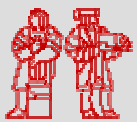
- You have all seen classes in your labs
- Basic anatomy
  - public class className{
    - Data members
    - Constructor
    - Methods

    }



Constructors

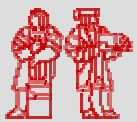
- Constructors provide objects with the data they need to initialize themselves, like “How to Assemble” instructions.
- Objects have a default constructor that takes no arguments, like LightSwitch().
- We can define our own constructors that take any number of arguments.
- Constructors have NO return type and must be named the same as the class:
  - ClassName(argument signature) { body }



Recall the LightSwitch Class

```
class LightSwitch {
    boolean on = true;
}
```

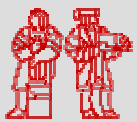
- The keyword class tells java that we're defining a new type of Object.
- Classes are a blueprint.
- Objects are instances of classes.
- Everything in Java (except primitives) are Objects and have a Class.



Using Objects

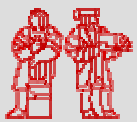
```
public static void main(String[] args) {
      LightSwitch s = new LightSwitch();
      System.out.println(s.isOn);
      s.flip();
      System.out.println(s.isOn);
}
```

- The new keyword creates a new object.
- new must be followed by a constructor.
- We call methods like:
  - variableName.methodName(arguments)



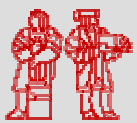
The LightSwitch Class

```
class LightSwitch {

  boolean on = true;

  boolean isOn() {
    return on;
  }

  void switch() {
    on = !on;
  }
}
```



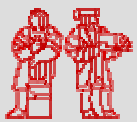
A Different LightSwitch Class

```
class LightSwitch {

  int on = 1;

  boolean isOn() {
    return on == 1;
  }

  void switch() {
    on = 1 - on;
  }
}
```



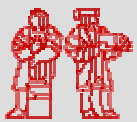
Abstraction

- Both LightSwitch classes behave the same.
- We treat LightSwitch as an **abstraction:** we do not care about the internal code of LightSwitch, only the external behavior
- Internal code = *implementation*
- External behavior = *interface*



Why is Abstraction Important?

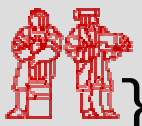
- We can continue to refine and improve the implementation of a class so long as the interface remains the same.
- All we need is the interface to an Object in order to use it, we do not need to know anything about how it performs its prescribed behavior.
- In large projects involving several teams, programmers only need to know what is necessary for their part of the code (eg. Microsoft, Google, Goldman Sachs, Morgan Stanley and other financial companies)



Breaking the Abstraction Barrier

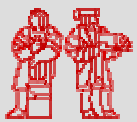
- A user of LightSwitch that relied on the boolean field would break if we changed to an integer field

```
class AbstractionBreaker {
  public static void main(String[] args) {
    LightSwitch ls = new LightSwitch();

    if (ls.on) // now broken!
      System.out.println("light is on");
    else
      System.out.println("light is off");
  }
}
```



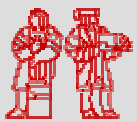
Public versus Private

- Label fields and methods **private** to ensure other classes can't access them
- Label fields and methods **public** to ensure other classes can access them.
- If they are not labeled public or private, for now consider them public.



A Better LightSwitch

```
class LightSwitch {

  private boolean on = true;

  public boolean isOn() {
    return on;
  }

  public void switch() {
    on = !on;
  }
}
```



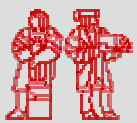
Enforcing the Abstraction Barrier

- By labeling the on field private . . .

```
class LightSwitch {
  private boolean on = true;

  // . . .
}
```

- Now AbstractionBreaker's attempt to access the on field would not have compiled to begin with.

```
if (ls.on)  // would never have compiled
```



Primitives vs Objects

- Two datatypes in Java: *primitives* and *objects*

- Primitives: byte, short, int, long, double, float, boolean, char

== tests if two primitives have the same value

- Objects: defined in Java classes

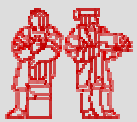
== tests if two objects are the same object

References

- The **new** keyword always constructs a new unique instance of a class
- When an instance is assigned to a variable, that variable is said to *hold a reference* or *point* to that object

```
Person g = new Person(“Mwangi", 21);
Person h = new Person(“Mwangi", 21);
```

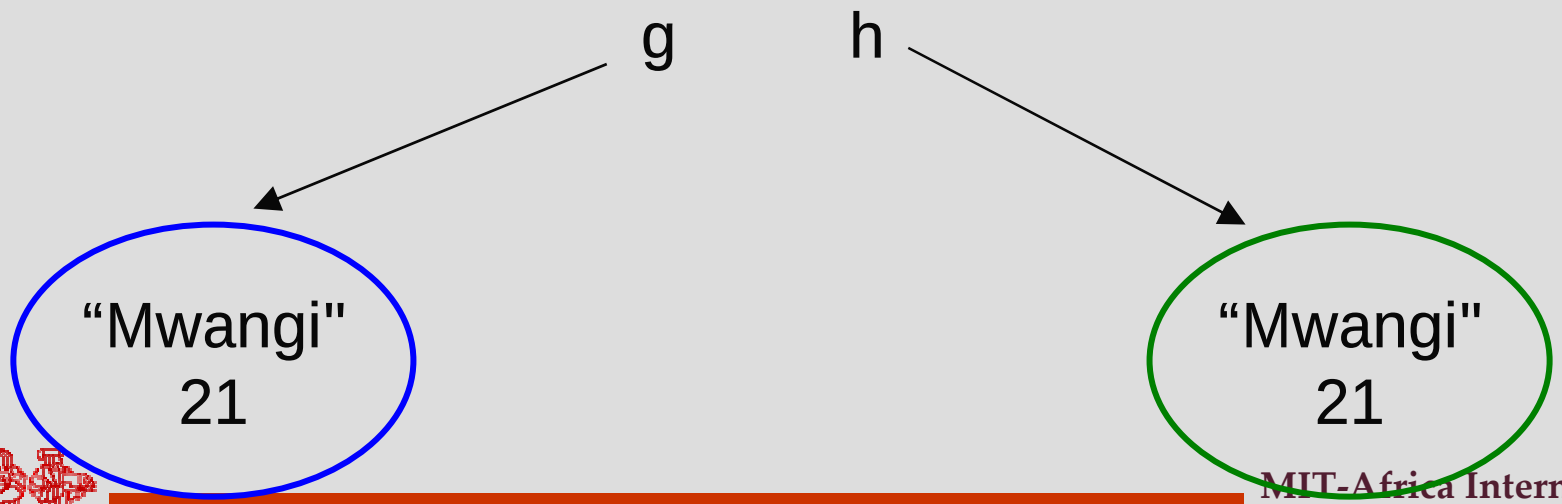
- g and h hold references to two different objects that happen to have identical state

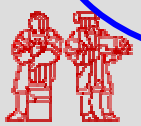


Reference Inequality

- g != h because g and h hold references to different objects

```
Person g = new Person(“Mwangi", 21);
Person h = new Person(“Mwangi", 21);
```

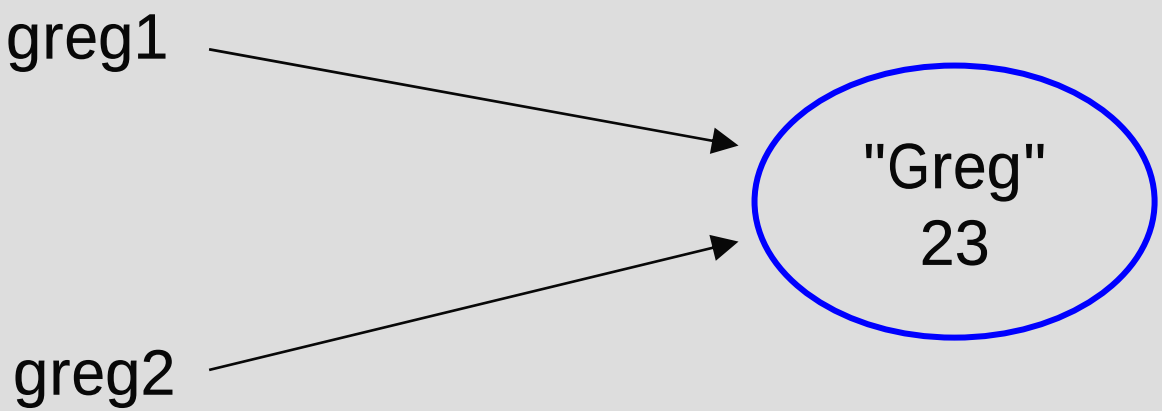


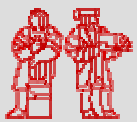


Reference Equality

- greg1 == greg2 because greg1 and greg2 hold references to the same object

```
Person greg1 = new Person("Greg", 23);
Person greg2 = greg1;
```





Equality Quiz 1

- Is `(a == b)`?

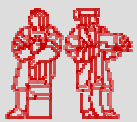
```
int a = 7;
int b = 7;
```

- Answer: Yes

- Is `(g == h)`?

```
Person g = new Person(“Mwangi", 21);
Person h = new Person(“Mwangi", 21);
```

- Answer: No



Equality Quiz 2

- true or false?

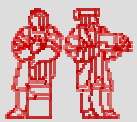
```
Person g = new Person(“James", 22);
Person h = new Person("James", 22);
Person lucy1 = new Person(“Lucy", 19);
Person lucy2 = lucy1;
```

a) g == h — false

b) g.getAge() == h.getAge() — true

c) lucy1 == lucy2 — true

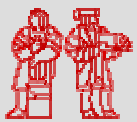
d) lucy1.getAge() == lucy2.getAge(); — true



Java API

- You can get information on all in-built Java classes/methods by browsing the Java Application Programming Interface (API)

- This documentation is essential to building any substantial Java application



EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005