



# Accelerating Information Technology Innovation

<http://aiti.mit.edu>

Cali, Colombia  
Summer 2012  
Lesson 06 – Classes and Objects

# What do we know so far?

---

- Primitives: int, float, double, boolean, char
- Variables: Stores values of one type.
- Arrays: Store many of the same type.
- Control Structures: If-then, For Loops.
- Methods: Block of code that we can pass arguments to and run multiple times.
- Is this all we want?

# Object-Oriented Programming

---

- Programming using *objects*
- An object represents an entity
  - Real world object: `String`, car, watch, ...
  - Abstract object: list, network connection, ...
- Objects have two parts:
  - **State**: Properties of an object.
  - **Behavior**: Things the object can do.

# Objects

---

- 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?

---

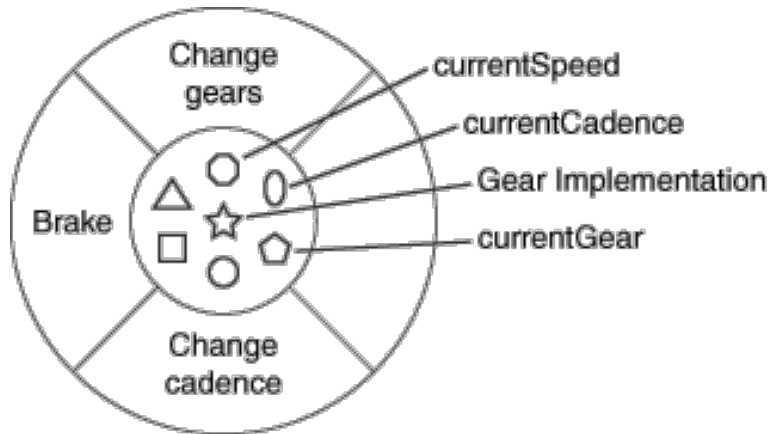
- **Modularity**: Once we define an object, we can reuse it for other applications.
- **Abstraction**: Programmers don't need to know exactly how the object works. Just the interface.
- **Encapsulation**: Hide the internal mechanisms to keep consistency.

# Abstraction

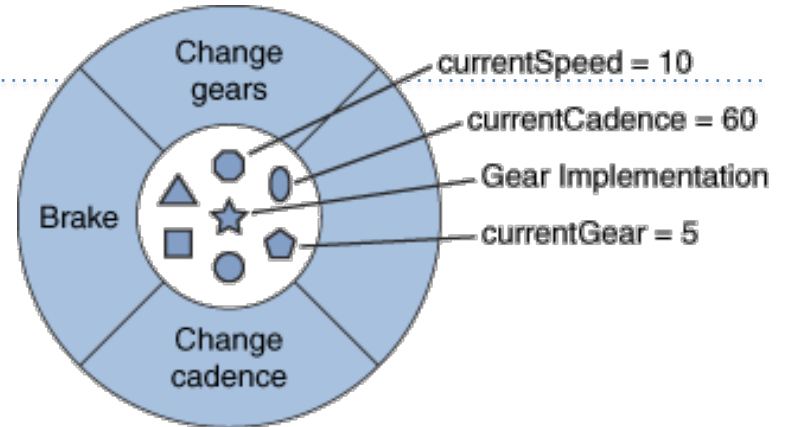
---

- We abstract away details to deal with complex problems.
  - Necessary for forming relationships between complex pieces of code.
  - The art is knowing which details to hide away and which to preserve.
  - What forms of abstraction have we seen so far?
- Example:
  - Different cars can use the same parts.
  - You don't need to know how an engine works in order to drive a car.

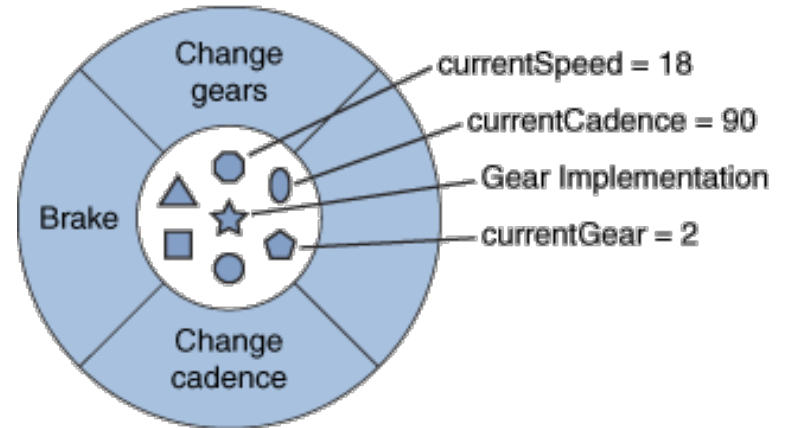
# Classes



A Bicycle Class



MyBike



YourBike

Two instances of the Bicycle Class

# Our first Class: LightSwitch

---

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

- What is the state of a LightSwitch?
- State stored in **fields**; here it's "isOn".
- Fields are accessed using:
  - variableName.fieldName
  - (We'll discuss other types of fields later)
- What are the behaviors of a LightSwitch?



# Our First Class: LightSwitch

---

```
class LightSwitch {  
}
```

- `class` keyword tells Java you are creating a class
- The class must reside in a file named *ClassName.java*
  - Ex: LightSwitch.java
- Currently, our class does nothing...

# Adding State

---

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

- What is the state of a LightSwitch?
- State stored in **fields**; here it's "isOn".
- Fields are accessed using:
  - variableName.fieldName
  - (We'll discuss other types of fields later)
- What are the behaviors of a LightSwitch?

# Adding Behavior

```
class LightSwitch {  
    boolean isOn = true;  
    void flip() {  
        this.isOn = !this.isOn;  
    }  
}
```

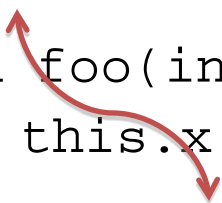
- We define methods in a class to add behavior
  - Methods change the state of the object and affect system state
- `this.isOn` accesses the `isOn` field.
- What behavior does `LightSwitch` have now?

# this Keyword

---

- Reference to the current object
  - The object whose method is being called
- Used to access fields:

```
class SimpleClass {  
    int x = 0;    //Field of SimpleClass  
  
    void foo(int x) {  
        this.x = x;  
    }  
}
```



# 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)`
- What does this code output?

# Constructors

---

- Constructors initialize the object after memory is allocated.
  - We can pass constructors data needed during initialization
- Objects have a default constructor that takes no arguments, like `LightSwitch()`

# Constructors

---

- We can define our own constructors that take any number of arguments.
  - `LightSwitch(boolean startState)`
  
- Constructors have NO return type and must be named the same as the class:
  - `ClassName(argument signature) { body }`

# Constructors

---

```
class LightSwitch {
    boolean isOn;
    void flip() {
        this.isOn = !this.isOn;
    }
    LightSwitch(boolean startState) {
        this.isOn = startState;
    }
}
```

- The LightSwitch() constructor no longer works. How do we [instantiate](#) an object?



# Multiple Constructors

---

- We can have multiple constructors.
- Constructors can call each other.

```
LightSwitch() {  
    this(true);  
}
```

```
LightSwitch(boolean startState) {  
    this.isOn = startState;  
}
```

# Review

---

- What two properties do objects have?
- What is the difference between a class and an object?
- What is a field?
- What does the **this** keyword mean?
- What does the **new** keyword do?
- What is a constructor?

# BankAccount Example

---

```
public class BankAccount {  
    double balance;  
    String name;  
    BankAccount(String name,  
                 double openBalance){  
        this.name = name;  
        this.balance = openBalance;  
    } // Continued next slide  
    ...  
}
```

# BankAccount Example

---

```
...
double deposit(double amount) {
    balance += amount;
    return balance;
}
boolean withdraw(double amount) {
    if (amount < balance) {
        balance -= amount;
        return true;
    } else return false;
}
} // End BankAccount Class
```