Abstraction

Abstraction

- **Abstraction** is the process of hiding certain details and showing only essential information to the user.
- Abstraction is the process of hiding the internal details of an application from the outer world. Abstraction is used to describe things in simple terms. It’s used to create a boundary between the application and the client programs.
- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Abstraction

ATM machine

Check balance

Withdraw cash

Deposit cash

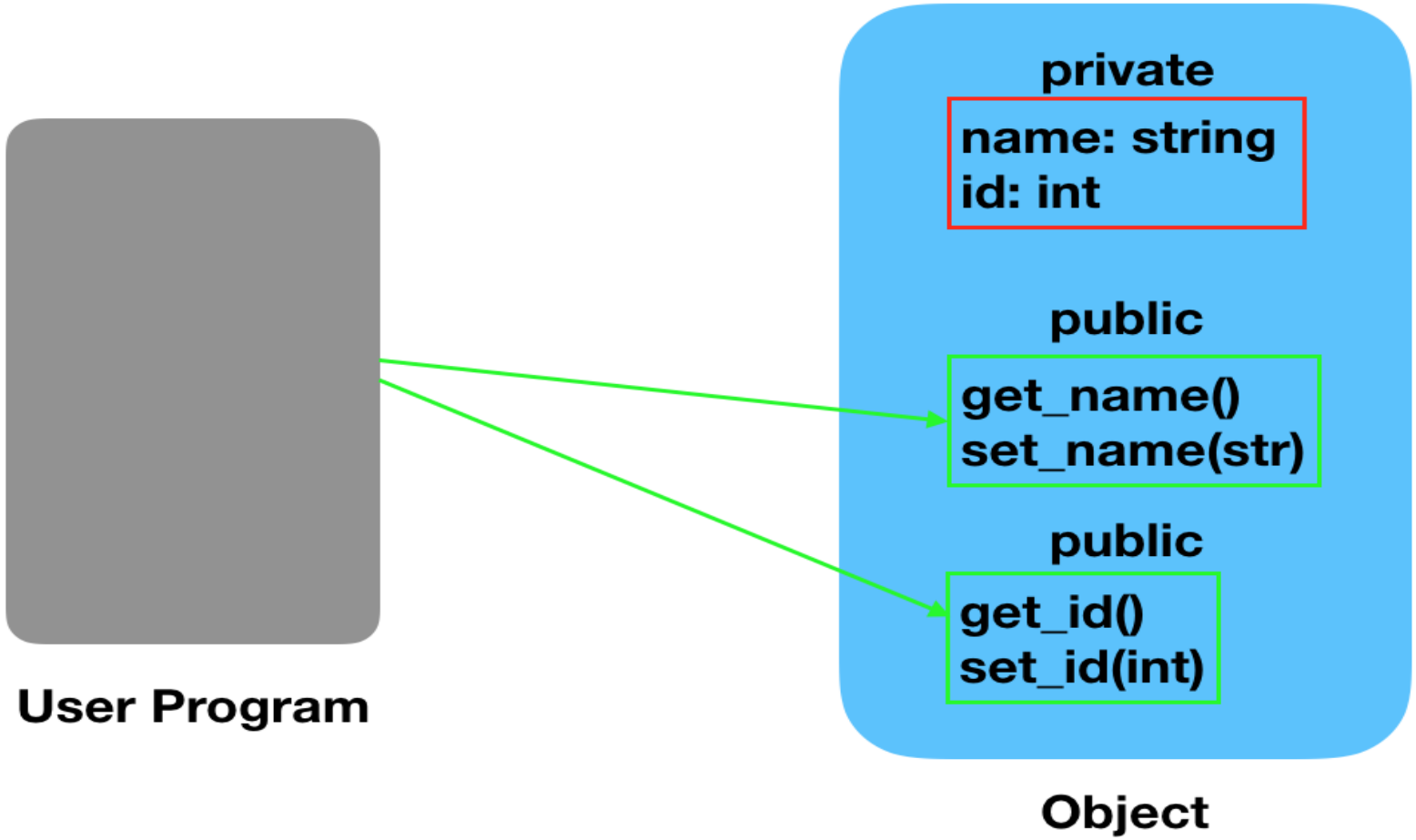
Print bill

Even though it performs a lot of actions
it doesn't show us the process
It has hidden its process by showing only the main things
like getting inputs and giving the output.

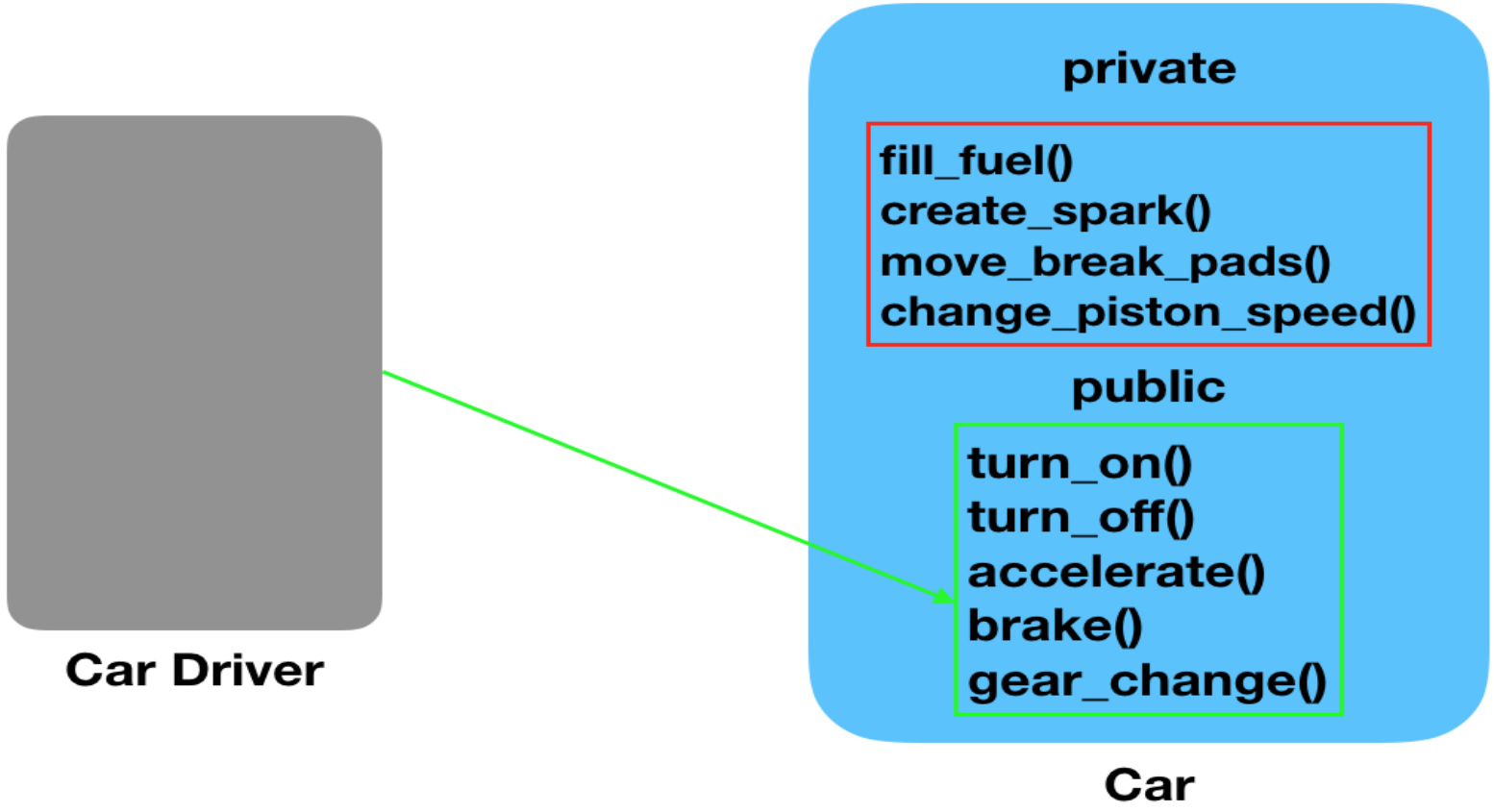
Types of Abstraction

- Data Abstraction:
  - When the object data is not visible to the outer world, it creates data abstraction. If needed, access to the Objects' data is provided through some methods.
- Process Abstraction:
  - We don't need to provide details about all the functions of an object. When we hide the internal implementation of the different functions involved in a user operation, it creates process abstraction.

Data Abstraction



Process Abstraction



In Java……!

```java
// Abstract class representing a Shape
abstract class Shape {
    // Abstract method to calculate area
    public abstract double calculateArea();
}

// Concrete class Circle extending Shape
class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    // Implementing the abstract method
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}

// Concrete class Rectangle extending Shape
class Rectangle extends Shape {
    private double length;
    private double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implementing the abstract method
    public double calculateArea() {
        return length * width;
    }
}

// Main class
public class Abstraction {
    Run | Debug
    public static void main(String[] args) {
        // Creating objects of concrete classes
        Circle circle_1 = new Circle(5.0);
        Rectangle rectangle_1 = new Rectangle(4.0, 6.0);

        // Using abstraction to calculate area
        System.out.println("Area of Circle: " + circle_1.calculateArea());
        System.out.println("Area of Rectangle: " + rectangle_1.calculateArea());
    }
}
```

In Python……!

```python
from abc import abstractmethod

# Abstract class representing a Shape
class Shape():
    # Abstract method to calculate area
    @abstractmethod
    def calculate_area(self):
        pass

# Concrete class Circle extending Shape
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    # Implementing the abstract method
    def calculate_area(self):
        return 3.14 * self.radius * self.radius

# Concrete class Rectangle extending Shape
class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    # Implementing the abstract method
    def calculate_area(self):
        return self.length * self.width

# Creating objects of concrete classes
circle = Circle(5.0)
rectangle = Rectangle(4.0, 6.0)

# Using abstraction to calculate area
print("Area of Circle:", circle.calculate_area())
print("Area of Rectangle:", rectangle.calculate_area())
```

Task.....!!!!!!

- Animal Abstraction:
  - Create an abstract class Animal with abstract methods makeSound() and eat().
  - Implement concrete classes Dog and Cat that extend the Animal class.
  - Implement the makeSound() and eat() methods for each animal.
  - Create objects of both classes and invoke their methods.
- Vehicle Abstraction:
  - Define an abstract class Vehicle with abstract methods start() and stop().
  - Create concrete classes Car and Motorcycle that extend the Vehicle class.
  - Implement the start() and stop() methods for each vehicle type.
  - Create objects of both classes and test their start and stop functionality.