Objects Round 2

AP Computer Science

Classes

Templates for objects

- A class is merely a plan or blueprint for a possible object
- It does not by itself create any objects
  - Class == Car
  - Object == Toyota Prius
- Like **`int`** is a type and **`34`** is an instance of the type **`int`**
- An object is the actual data you can use in your code

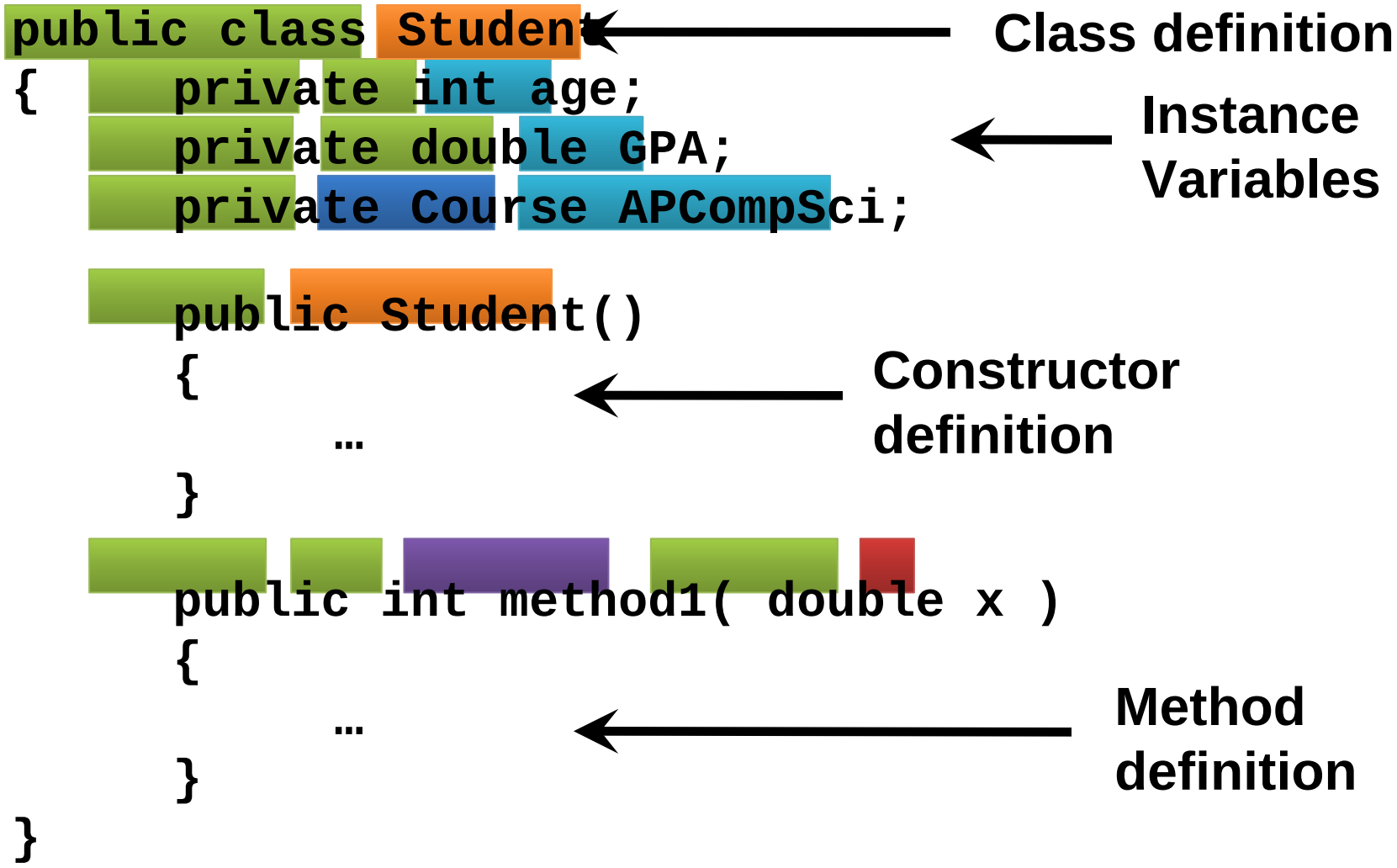
Templates for MANY objects

- Since the class is only a template it can be used to create many objects of the same type
- Realize that creating a class is really creating a new data type you can use in your program
- Each object created from a class has its own set of data and methods

Contain members and methods

- When designing classes, they contain two kinds of elements:
  - Members (instance variables) - the data describing the objects (color, age, location)
  - Methods - actions either the object can do or that can be done to the object (move, change color, get age)

Anatomy of a class definition



Members

Members are data inside an object

- Members (instance variables) are the actual data inside an object
- They can be primitive types or other object types
- They are hidden (**`private`**) from the outside world

```
public class Point
{
    private double x; // instance variable
    private double y; // instance variable
```

Declaring instance variables

- Please note that instance variables are only declared at the top of the class
- The assignment happens inside the constructor
- This is a common error of beginners

```
public class Point
{
    private double x; // instance
variable
    private double y; // instance
variable
```

Data visibility

- What do **`private`** and **`public`** mean?
- These keywords allow you to specify the scope or permissions of members and methods
- **`private`** means only methods from the same class can access an item
- **`public`** means any method can access the item

Methods

Methods are ways to interact with objects

- Methods allow you to do things
- Object methods usually allow you to manipulate the members (instance variables)
- They are usually visible (**`public`**) to the outside world
- Methods can be static or non-static
- Only non-static methods can interact with the members of an object

Static vs non-static methods

- Static methods can be executed without creating an object
- To call a static method you use class name dot method name
- Non-static methods do not exist before you create an object of the class
- To call a non-static method you use the reference variable dot method name

Accessor methods

- Because members are **`private`**, it is common to use methods to find out what values they are storing
- A method that *only returns the value of a instance variable* is called an **accessor method**

```
public double getX() //accessor for x
{
   return x;
}

public double getY() //accessor for y
{
  return y;
}
```

toString() method

- The toString() method is a very common method for us to use
- toString() will return the values for all of the instance variables for a particular object
- You call toString() by placing the reference variable name inside a print() or println() statement

```
public String toString() //toString() in Point
{
  return "" + x + " " + y;
}
```

toString() example

```
//main method
Point p = new Point(5, 10);
System.out.println(p);
```

```
public class Point {
  private double x;
  private double y;

  public Point(double xPos, double yPos) {
    x = xPos;
    y = yPos;
  }

  public String toString() {
    return "" + x + " " + y;
  }
}
```

Output
5 10

Modifier methods

- Because members are **`private`**, it is common to use methods to change their values
- A method that *only changes the value of a instance variable* is called a **modifier method**

```
public void setX(double newX) //modifier for x
{
  x = newX;
}

public void setY(double newY) //modifier for y
{
  y = newY;
}
```

Modifier & accessor methods

- **Accessor** methods normally start with **get**
- **Modifier** methods normally start with **set**
- This is not required but does make it easy to find modifier and accessor methods in your program

```
public double getX() //accessor for x
{
  return x;
}

public void setY(double newY) //modifier for y
{
  y = newY;
}
```

Variable Scope

Instance variables

- Instance variables are the data associated with each object
- Instance variables are declared as private
- Instance variables are available to every non-static method in the class

```
public class Student
{
  private int age;  //instance variable
  private double GPA;  //instance variable

  //rest of Student class.....
}
```

Local variables

- Local variables will either consist of the parameters you pass into a method, or variables you declare inside a method
- Local variables are only available inside the method in which they are declared

```
//ints a & b are both local variables
public void addNums(int a, int b)
{
  //int answer is a local variable
  int answer = 0;
  answer = a + b;
  return answer;
}
```

Variable scope

- The scope of a variable refers to the time when it is available for use
- For local variables the scope is the method the variable is defined in
- For instance variables the scope is the entire class

```
//the scope of a and b is the method addNums
public void addNums(int a, int b)
{
  return a + b;
}
```

Variable scope

```
public class Student {

    //instance variable
    private int age;

    //default constructor
    public Student() {
        age = 0;
    }

    //modifier method
  public void setAge(int a) {

        //a is a local variable availabe in
setAge
        age = a;
    }
```

age scope

a scope

Equivalence testing

How do you tell if two objects are the same?

- If you have two primitive variables, you use the **==** operator
- However, with objects, this will only give you back **`true`** if the two references are pointing at exactly the same object
- Sometimes this is what you want to know, but objects can be equivalent in other ways

Equivalence confusion

```
String s1 = new String("identical");
String s2 = new String("identical");
if( s1 == s2 )
  System.out.println("Same!");
else
  System.out.println("Different!");
if( s1.equals( s2 ) )
  System.out.println("Same!");
else
  System.out.println("Different!");
```

- In this example, the **==** operator will say they are different, but the **`equals()`** method will say that they are the same
- Every object has an **`equals()`** method