Objects and classes

- Objects:
  - An object is a composite piece of data which can react to messages sent to it.
  - The data type of an object is a class

- Classes:
  - A class is a data type
  - A class is like the "blueprint" of a set of objects
  - Classes have have *attributes* and *methods* (to describe the structure and behaviour if its objects.)
    * Attributes: variables describing the characteristics of objects in the class.
    * Methods: operations on objects of the class; how to react to messages from other objects.

Objects are not classes

- A class is a data type. An object is a particular value whose type is some class.
- An *object* is an *instance* of a class.
- An object has its own separate identity and its own separate state.
- The *state* of an object is the values currently assigned to its attributes.
- Each object is stored in different memory locations.
- Therefore the class definition does not describe doing something to a specific object, but rather describes its structure and how would object of the class react if they are sent messages (i.e. if someone applies an operation on the object.)

Classes and methods

- A class by itself ***does not*** create any objects.
- Objects (or instances) are created in some other class by using the `new` operator.
- A method by itself is not executed.
- Methods are invoked (or called) from some other class.

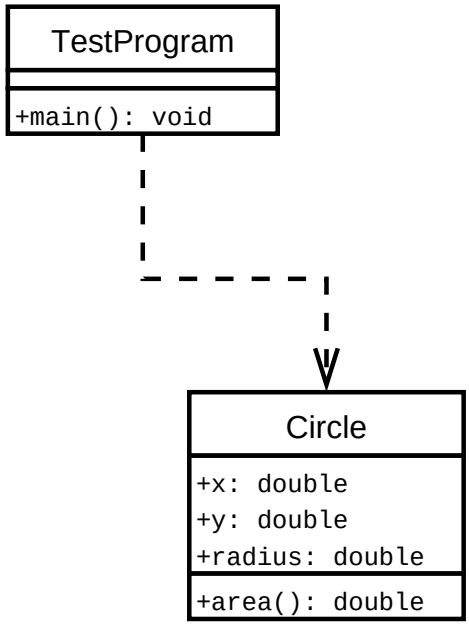
```
public class Circle
{
  double x, y, radius;

  double area()
  {
    return Math.PI * radius * radius;
  }
}
```

Classes and methods

- A *client* of a class is whomever uses the class.

```
public class TestProgram
{
  public static void main(String[] args)
  {
    double a;
    Circle c = new Circle();
    c.x = 2.0;
    c.y = -3.0;
    c.radius = 4.0;
    a = c.area();
  }
}
```

Classes and methods



Classes and methods

- The client of a class does not need to be the class with the `main` method.

```
public class AreaCalculator
{
  double compute_area()
  {
    double a;
    Circle c = new Circle();
    c.x = 2.0;
    c.y = -3.0;
    c.radius = 4.0;
    a = c.area();
    return a;
  }
}
```

Classes and methods

- The client of a class does not need to be the class with the `main` method.

```
public class AnotherTestProgram
{
  public static void main(String args)
  {
    double a;
    AreaCalculator q;
    q = new AreaCalculator();
    a = q.compute_area();
  }
}
```

Classes and methods

- A class may have more than one client
- It doesn't matter if they have methods or attributes with the same names.

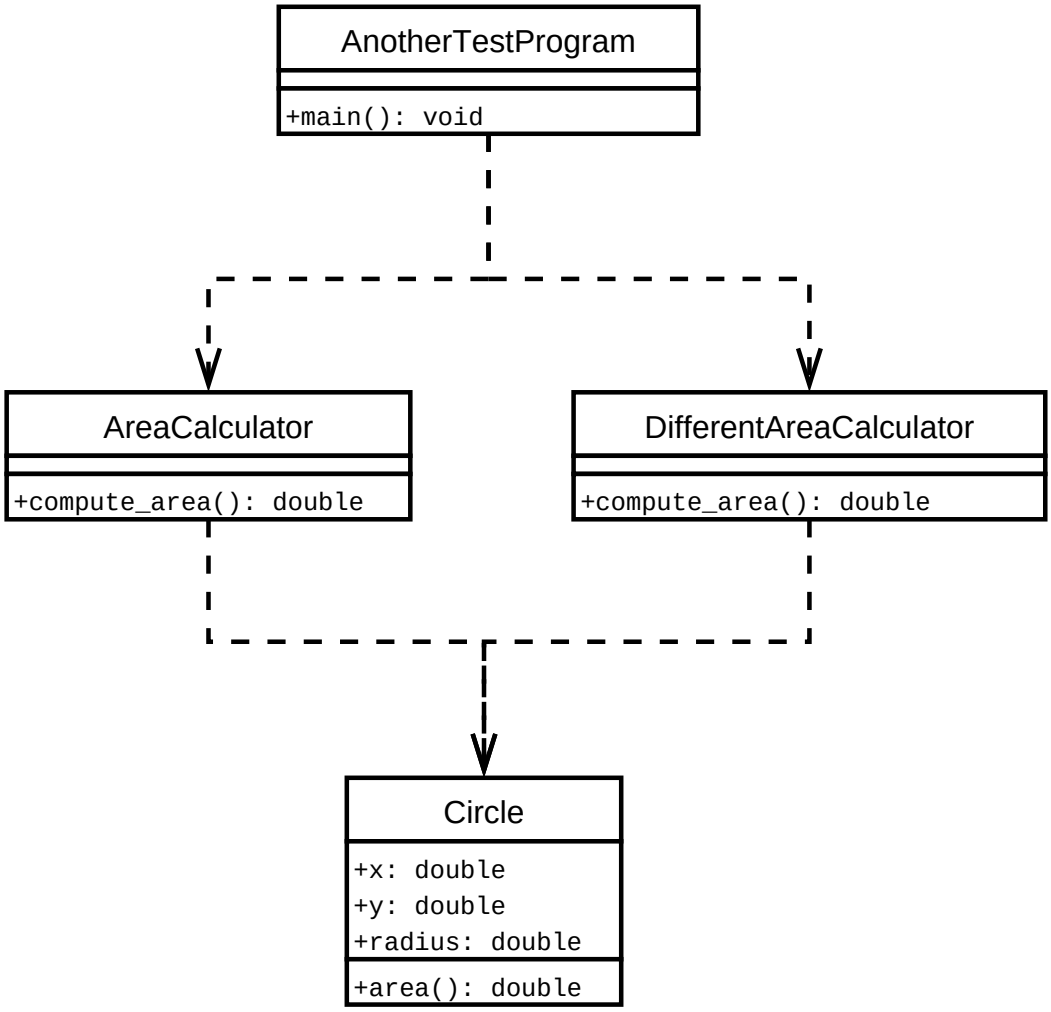
```
public class DifferentAreaCalculator
{
  double compute_area()
  {
    double a;
    Circle c = new Circle();
    c.x = 0.0;
    c.y = 0.0;
    c.radius = 2.0;
    a = c.area();
    return a;
  }
}
```

Classes and methods

- A class may have more than one client

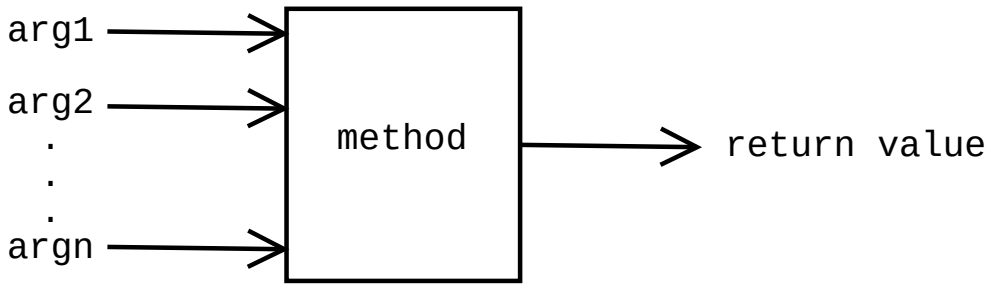
```
public class AnotherTestProgram
{
  public static void main(String args)
  {
    double a, b;
    AreaCalculator q;
    DifferentAreaCalculator p;
    q = new AreaCalculator();
    p = new DifferentAreaCalculator();
    a = q.compute_area();
    b = p.compute_area();
  }
}
```

Classes and methods



Methods as functions

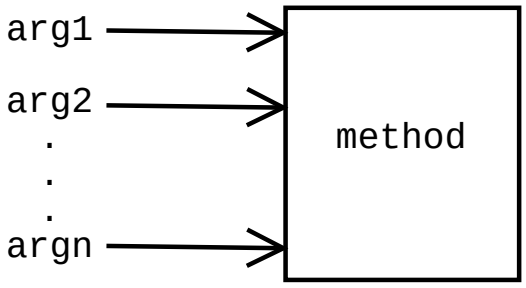
- Methods can be viewed as a "black box" with inputs and outputs:



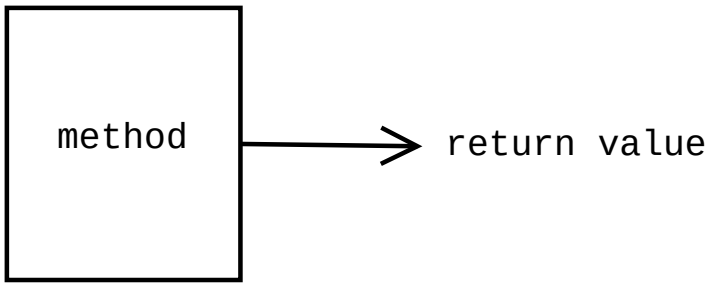
- There are three kinds of methods:
  - Mutators: Modify the state of objects,
  - Accessors: Return information about the object,
  - Constructors: Initialize a newly created object.

Method types

- Mutators are usually `void` methods, which do not return anything, but modify the state of the object:



- Accessor methods may only return values without expecting any arguments as input:



Constructors

- Special methods, whose syntax is given by

```
class_name(list_of_arguments)
{
    statements;
}
```

- For example:

```
public class Student {
    String name;
    long id;
    String program;
    String faculty;

    Student(String n, long i)
    {
        name = n;
        id = i;
    }
    //...
}
```

Constructors (contd.)

- A constructor method gets executed when a new object of the class gets created using the new keyword. Therefore, the general syntax for the expression used to create objects is:

```
new class_name(list_of_actual_arguments);
```

- For example

```
Student al;
al = new Student(“Alan Turing”, 110011223331);
```

Constructors

```
public class Circle {
  double x, y, radius;

  Circle(double x0, y0, r)
  {
    x = x0;
    y = y0;
    radius = r;
  }

  double area()
  {
    return Math.PI * radius * radius;
  }
}
```

Method types

```
public class Circle {
  double x, y, radius;

  Circle(double x0, y0, r)
  {
    x = x0;
    y = y0;
    radius = r;
  }

  double area()
  {
    return Math.PI * radius * radius;
  }

  // continues...
```

Method types

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

double getY()   // accessor
{
  return y;
}

double getRadius()  //accessor
{
  return radius;
}

//continues ...
```

Method types

```
  void setX(double x1)  // mutator
  {
    x = x1;
  }

  void setY(double y1)  // mutator
  {
    y = y1;
  }

  void setRadius(double r)  // mutator
  {
    radius = r;
  }
}
```

Method types

```
public class AreaCalculator
{
  double compute_area()
  {
    double a;
    Circle c = new Circle(0.0,-4.0, 7.0);
    c.setX(2.0);
    c.setY(-3.0);
    c.setRadius(4.0);
    a = c.area();
    return a;
  }
}
```

Monsters

- Develop a representation for "Monsters" in a video game, where the monsters have:
  - a position,
  - a number of "hitpoints" between 0 and 100, representing its health. If the number reaches 0, the monster is dead.
- And, a monster can:
  - receive damage when attacked.

Monsters

```
class Monster {
  double x, y;
  int hp = 100;
  boolean alive = true;

  void get_damaged()
  {
    if (hp > 0) hp = hp - 10;
    if (hp <= 0) alive = false;
  }
}
```

Monsters

```
class Monster {
  double x, y;
  int hp;
  boolean alive;

  Monster(double x0, double y0)
  {
    x = x0;
    y = y0;
    hp = 100;
    alive = true;
  }

  void get_damaged()   // mutator
  {
    if (hp > 0) hp = hp - 10;
    if (hp <= 0) alive = false;
  }

  // continues
```

Monsters

```
  boolean isAlive()  // accessor
  {
    return alive;
  }

  double getX() { return x; }

  double getY() { return y; }

  int hitPoints()
  {
    return hp;
  }
}
```

Monsters

```
class Monster {
  double x, y;
  int hp;
  boolean alive;

  Monster(double x0, double y0)
  {
    x = x0;
    y = y0;
    hp = 100;
    alive = true;
  }

  void get_damaged()   // mutator
  {
    if (hp > 0) hp = hp - 10;
    if (hp <= 0) alive = false;
  }

  // continues
```

Monsters

```
public class Game {
  public static void main(String[] args)
  {
    Monster ernesto, yannick;
    ernesto = new Monster(0.0, 0.0);
    yannick = new Monster(50.0, -30.0);
    yannick.get_damaged();
    int i = 1;
    while (i <= 10) {
      ernesto.get_damaged();
      i++;
    }
    System.out.println(yannick.isAlive());
    System.out.println(ernesto.isAlive());
  }
}
```

Monsters

```
  void recover()
  {
    if (alive && hp < 100)
    {
      hp = hp + 5;
    }
  }

  void attack(Monster other)
  {
    if (alive) {
      other.get_damaged();
    }
  }
}
```

Monsters

```
  void recover()
  {
    if (alive && hp < 100)
    {
      hp = hp + 5;
    }
  }

  void attack(Monster other)
  {
    if (this.alive) {
      other.get_damaged();
    }
  }
}
```

Monsters

```
public class Game {
  public static void main(String[] args)
  {
    Monster ernesto, yannick;
    ernesto = new Monster(0.0, 0.0);
    yannick = new Monster(50.0, -30.0);
    yannick.get_damaged();
    int i = 1;
    while (i <= 10) {
      ernesto.get_damaged();
      ernesto.attack(yannick);
      i++;
    }
    System.out.println(yannick.isAlive());
    System.out.println(ernesto.isAlive());
  }
}
```

The End