清森学校
BEIJING QINGSEN SCHOOL

AP-CSA Heritance

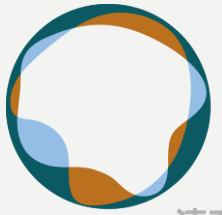
YING HUANG

DEC.2022

Object-oriented Programming

Object-oriented programming has three main features:

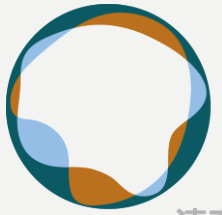
- **Objects** have data (fields) and behavior (methods) and do the work in an object-oriented program.
- **Inheritance** allows for cleaner code since a class can inherit fields and behavior from another class instead of copying code from class to class.
- **Polymorphism** allows for specialized behavior based on the run-time type.



REVIEW
CLASS AND OBJECT

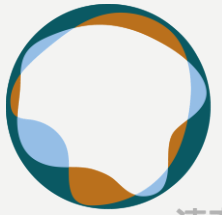
All objects created by the same class have the same **fields** and **methods**.

- A **field** is something the object knows about itself
- A **method** is a thing the object can do.



EXAMPLE
CLASS AND OBJECT

See IDEA



清森学校
BEIJING QINGSEN SCHOOL

INHERITANCE

INHERITANCE 继承

You may have heard of someone coming into an inheritance, which often means they were left something from a relative who died. Or, you might hear someone say that they have inherited musical ability from a parent.

In Java, all classes can **inherit** attributes (instance variables) and behaviors (methods) from another class.

The class being inherited from is called the **parent class or superclass**. The class that is inheriting is called the **child class** or **subclass**.

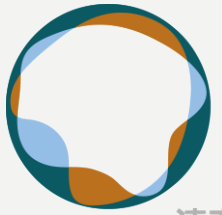
INHERITANCE 继承

Inheritance: A way to form new classes based on existing classes, taking on their attributes/behavior.

- a way to group related classes
- a way to share code between two or more classes

One class can extend another, absorbing its data/behavior.

- **Superclass:** The parent class that is being extended.
- **Subclass:** The child class that extends the superclass and inherits its behavior.
  - Subclass gets a copy of every instance variable and method from superclass



INHERITANCE 继承

- A **UML (Unified Modeling Language) class diagram** shows classes and the relationships between the classes as seen in Figure 1.

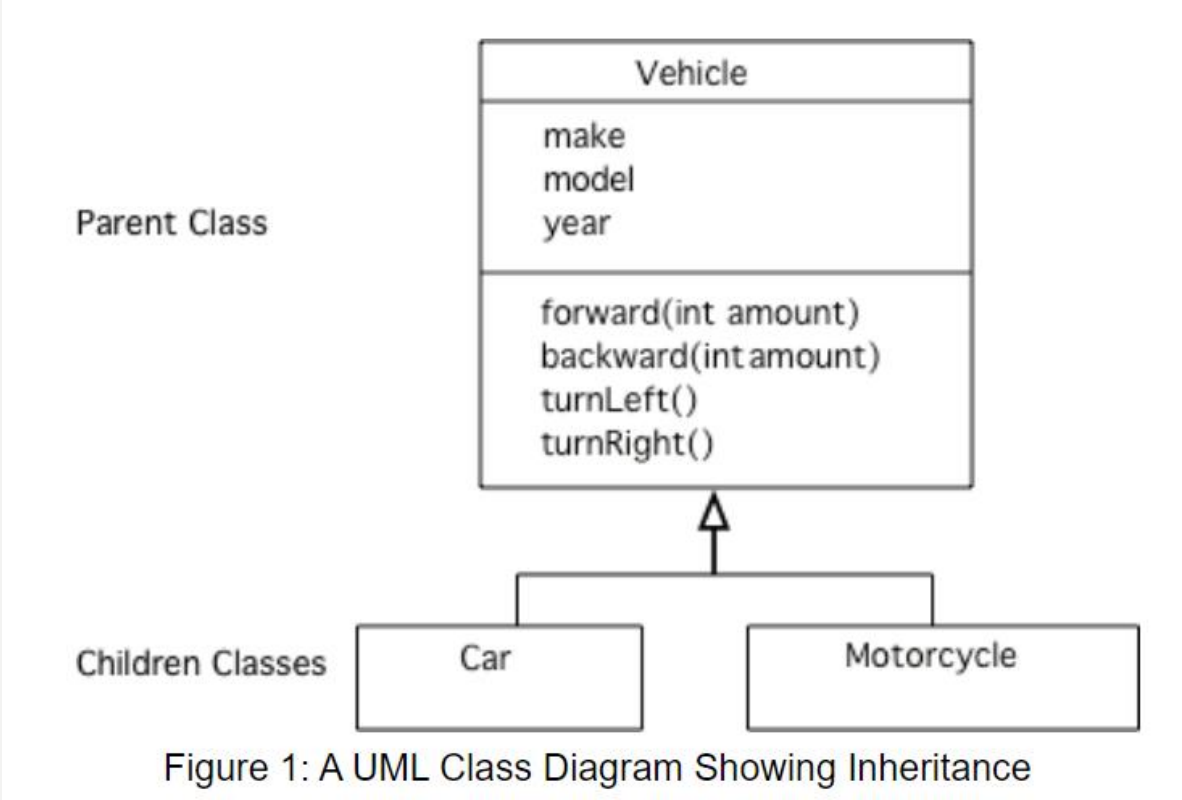


Figure 1: A UML Class Diagram Showing Inheritance

INHERITANCE 继承

- Syntax

A class can extend another class by using the keyword **extends** then the name of the class it is extending.

Below, Car extends Vehicle.

```
public class Car extends Vehicle{
    // not show
}
```

If you leave off the extends keyword when you declare a class then the class will inherit from the **Object class**. The Vehicle class declared below will inherit from the **Object class.**

```
public class Vehicle {
    private String model;
    private String make;
    private int year;

    public void forward(int amount){}
    public void backward(int amount){}
    public void turnLeft(){}
    public void turnRight(){}

}
```

INHERITANCE 继承

- "has a" relationship: represent instance variables in our class

  Example:

  A Vehicle class could have model, year as instance variable.

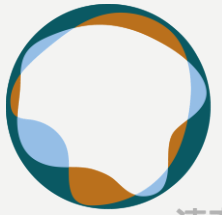
  A Person class could have name and birthday as instance variable.

- "is a" relationship: one class is a more specific example of another class.

  Example:

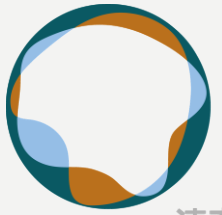
  a Car is a kind of Vehicle

  a Student is a Pearson



INHERITANCE 继承

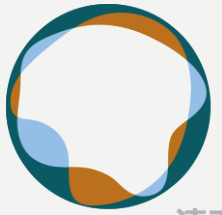
- If you notice that several classes share the same data and/or behavior, you can pull that out into a parent class. This is called **generalization**.
- Inheritance allows you to reuse data and behavior from the parent class.
- Conversely, inheritance is also useful for **specialization** which is when you want most of the behavior of a parent class, but want to do at least one thing differently and/or add more data.



INHERITANCE 继承

Code

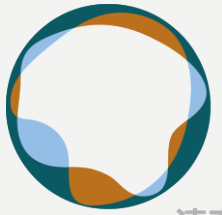
See IDAE



清森学校
BEIJING QINGSEN SCHOOL

INHERITANCE 继承

- We can create a hierarchy of classes that allow us to reuse common attributes and behaviors.
- A subclass uses the IS A relationship signifying that it is a more specific example of the broader superclass.
- The keyword extends is used to establish the relationship between a superclass and the subclass.
- A class can only extend one superclass.

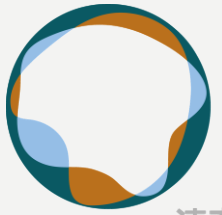


INHERITANCE 继承

- Subclasses **inherit all the public and private instance variables** in a superclass that they extend, but they **cannot directly access private variables.**

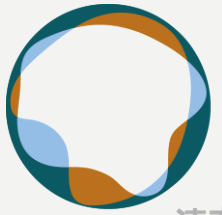
- And **constructors are not inherited**.

- How do you initialize inherited private variables if you don't have direct access to them in the subclass?



CONSTRUCTOR

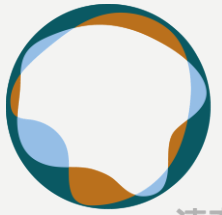
- Since constructors are not inherited, a subclass needs to create its own constructor.
- Subclass constructors **must call the parent constructor**.
- The superclass constructor can be called from **the first line of a subclass constructor** by using the keyword **super** and passing appropriate parameters.
- When no superclass is defined, the **Object class** is the superclass



INHERITANCE 继承

Code

See IDAE



清森学校
BEIJING QINGSEN SCHOOL

REVIEW

What is the class?

ANS:

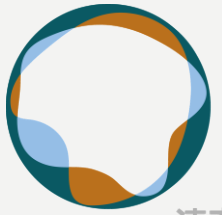
A class is the formal implementation, or blueprint, of the attributes and behaviors of an object.

What is the superclass and subclass ?

ANS:

Superclass: The parent class that is being extended.

Subclass: The child class that extends the superclass and inherits its behavior.



SUPERCLASS VS. SUBCLASS

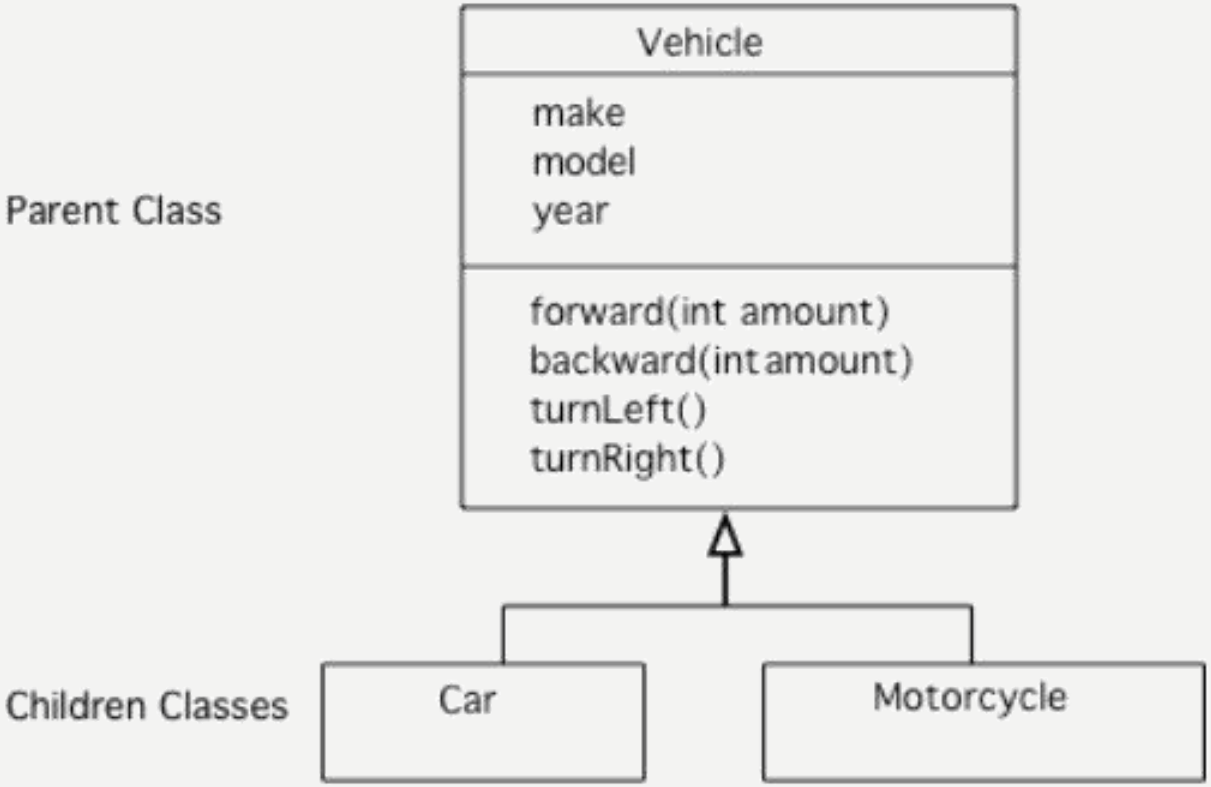
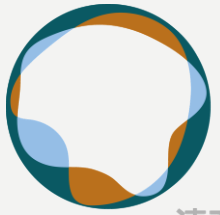


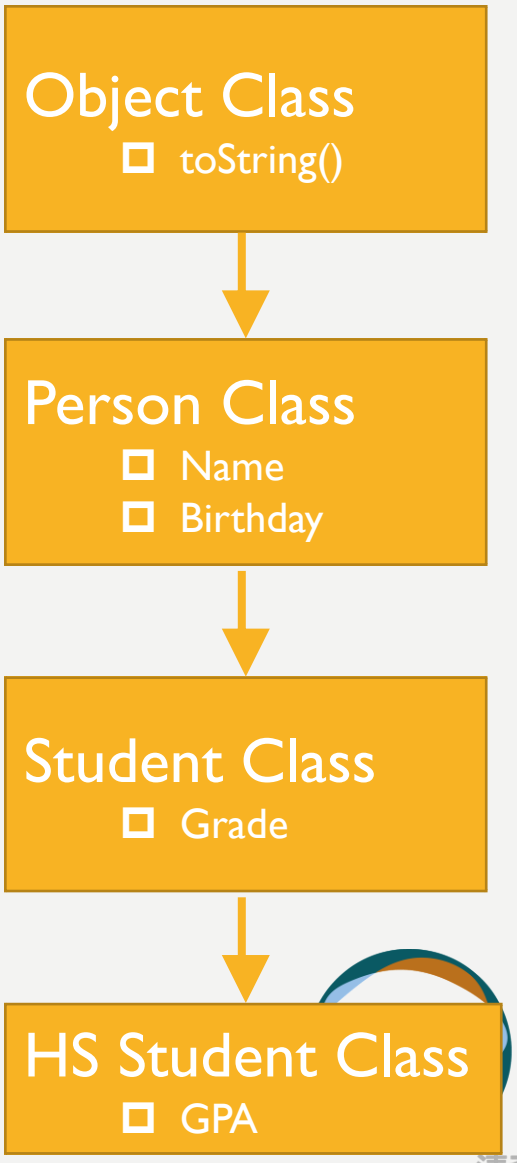
Figure 1: A UML Class Diagram Showing Inheritance



CLASS HIERARCHY

As we extend down, the subclasses have access to all of the public methods of the parent.

For example, if the Person class has a public getName() method, any Person, Student, or HS Student Class object could call that method



REVIEW

What Java keyword is used to specify the parent class?

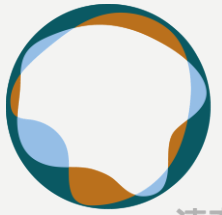
A. superclass

B. parent

C. extends

D. Class

ANS：C



REVIEW

A bookstore is working on an on-line ordering system. For each type of published material (books and movies) they need to track the id, title, author(s), date published, and price. Which of the following would be the best design?

A. Create one class PublishedMaterial with the requested fields plus type

B. Create classes Book and Movie and each class has the requested fields

C. Create the class PublishedMaterial and have Book and Movie inherit from it all the listed fields

D. Create one class BookStore with the requested fields plus type

ANS： C

REVIEW

Given the class definitions of Point2D and Point3D below, which of the constructors that follow (labeled I, II, and III) would be valid in the Point3D class?

```
class Point2D {
   public int x;
   public int y;

   public Point2D() {}

   public Point2D(int x,int y) {
     this.x = x;
     this.y = y;
   }
   // other methods
}

public class Point3D extends Point2D
{
   public int z;

   // other code
}
```

```
// possible constructors for Point3D
I.  public Point3D() {}
II. public Point3D(int x, int y, int z)
    {
       super(x,y);
       this.z = z;
    }
III. public Point3D(int x, int y)
     {
        this.x = x;
        this.y = y;
        this.z = 0;
     }
```

A. II only
B. III only
C. I and II only
D. I, II, and III

ANS: D

OVERRIDING VS. OVERLOADING

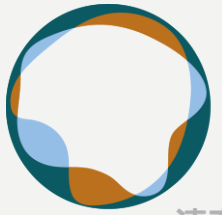
OVERRIDING

Override: To write a new version of a method in a subclass that replaces the superclass's version. To override an inherited method, the method in the child class must have the same signature

1. **Same method name,**
2. **Same parameter list (order, type)**
3. **Same return type (or a subclass of the return type)**

Have we done this before?

Answer: toString()



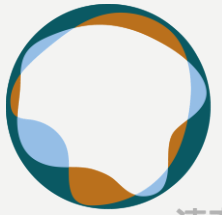
OVERRIDING - CODE

Superclass

```
public class Employee {
// some constructors and methods not shown
     public String getVacationForm() {
            return "pink";
      }
}
```

Subclass

```
public class Lawyer extends Employee {
// overrides getVacationForm method in Employee class
     public String getVacationForm() {
            return "yellow";
      }
}
```



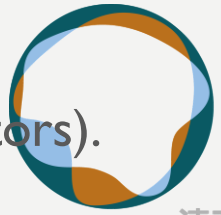
KEYWORD "SUPER"

Sometimes you want the subclass to do more than what a superclass' method is doing. You want to still execute the superclass method, but you also want to override the method to do something else.

But, since you have overridden the parent method how can you still call it? You can use super.method() to force the parent's method to be called.

We've used super() before to call the superclass' constructor. There are two uses of the keyword super:

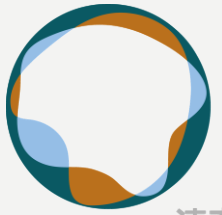
- **super();** or **super(arguments);** calls just the super constructor if put in as the first line of a subclass constructor.
- **super.method();** calls a superclass' method (not constructors).



OVERRIDING

Code

See IDAE



清森学校
BEIJING QINGSEN SCHOOL

KEYWORD "SUPER"

Example 1

Given the following class declarations, and assuming that the following declaration appears in a client program: Base b = new Derived();, what is the result of the call b.methodOne();?

ANS: ABDC

```
public class Base
{
   public void methodOne()
   {
     System.out.print("A");
     methodTwo();
   }

   public void methodTwo()
   {
     System.out.print("B");
   }
}
```

```
public class Derived extends Base
{
   public void methodOne()
   {
      super.methodOne();
      System.out.print("C");
   }

   public void methodTwo()
   {
     super.methodTwo();
     System.out.print("D");
   }
}
```

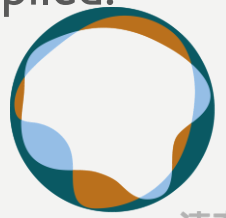
OVERRIDING

Remember that an object always keeps a reference to the class that created it and always looks for a method during execution starting in the class that created it.

If it finds the method in the class that created it, it will execute that method.

If it doesn't find it in the class that created it, it will look at the parent of that class.

It will keep looking up the ancestor chain until it finds the method. The method has to be there, or else the code would not have compiled.



OVERLOADING

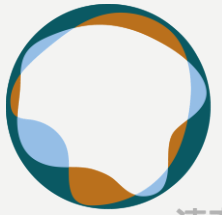
Methods are said to be **overloaded** when there are multiple methods with the same name but a different signature in the same class.

The methods are distinguished by:

1. Number of parameters
2. Type of the parameters
3. Order of the parameters

Have we done this before?

Answer: more constructors!



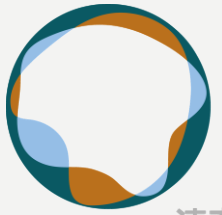
OVERLOADING - CODE

```
public class Overload{
        public void method1(int c)
        {…}

        public void method1(int c, double d)
        {…}

        public void method1(double c)
        {…}

        public void method1(double d, int c)
        {…}

}
```

Number of Parameters

Type of Parameters

Order of Parameters

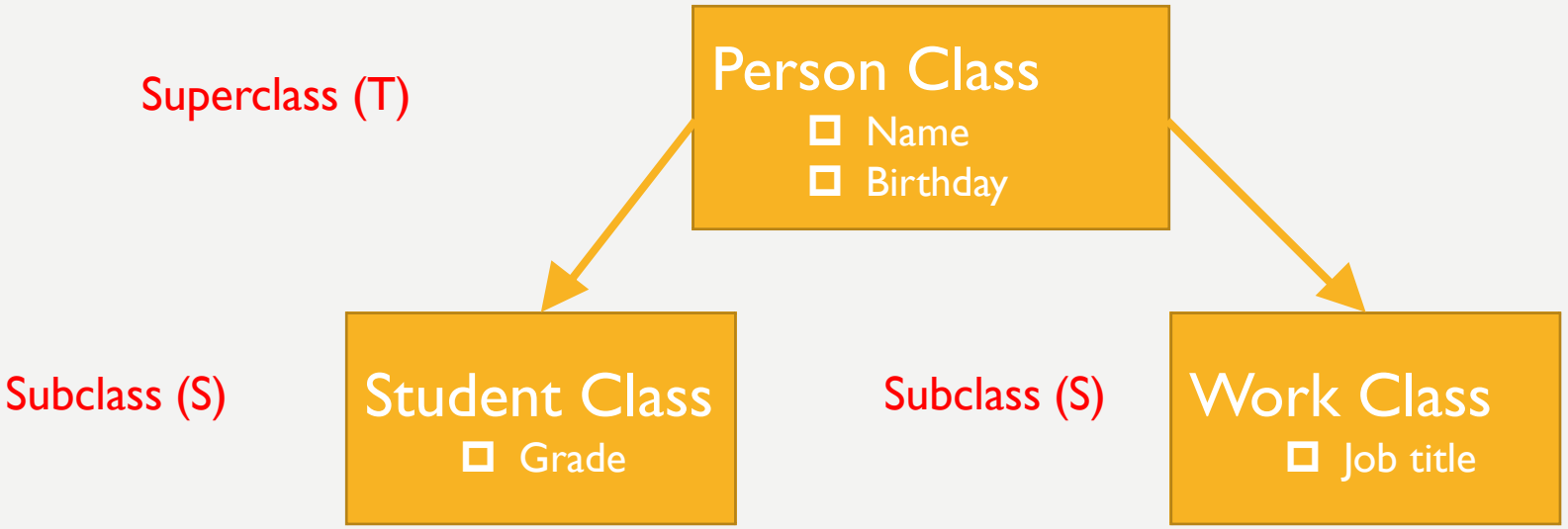


SUMMARY

Overriding	Overloading
Implementing Runtime Polymorphism	Implementing Compile time polymorphism
Occurring between superclass and subclass	Occurring the methods in the same class
The same signature i.e. same method name, method arguments and return type.	The names are the same but the parameters are different.

POLYMORPHISM

CLASS HIERARCHY



Class hierarchy facilities code reuse by putting common attributes and behaviors in the superclass.

In Java, we can say that when a class S is a class T , and S is a subclass to the T superclass.

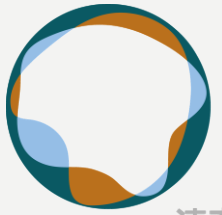
POLYMORPHISM 多态

- **Polymorphism** comes from the Greek poly meaning many and morph meaning forms.
- **Polymorphism** is the capability of a method to do different things depending on the object it is acting upon.
- In Java and other OOP languages, the concept of **polymorphism** means that an object can take on different forms depending on its implementation.
- Java can call the correct method even when an object is disguised as a more generic reference type.
  - method overriding(**run-time polymorphism**)
  - method overloading(**compile-time polymorphism**)

POLYMORPHISM 多态

Code

See IDAE



清森学校
BEIJING QINGSEN SCHOOL

POLYMORPHISM 多态

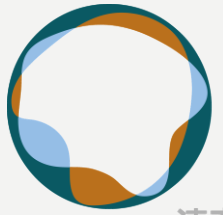
```
Person p1 = new Person("Cindy");
```

Reference Type
Variable declaration

Object Type
Variable instantiation

```
Person p2 = new Student("Alice","3333@qq.com",3.5);
```

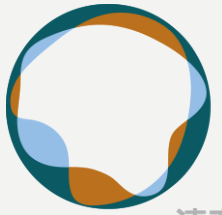
When we use a Superclass as a reference type T, then we can create an object as either the Superclass T, or any Subclass S.



POLYMORPHISM 多态

Code

See IDAE



清森学校
BEIJING QINGSEN SCHOOL

POLYMORPHISM 多态

Polymorphism allows flexibility when we create with a Superclass reference type.

- We can use a type T as a formal parameter in a method, then we pass any object of type T or S.
  - Syntax:

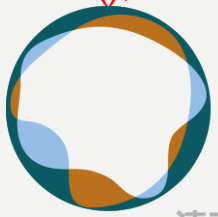
    public return type methodname (typeT object){…}

    Call the method: methodname(T/S);

    // using object T or object S to call this method.
- We can create Arrays and Arraylist of a T and store any type T or S objects.
  - Syntax:
    - **List:** TypeT [] name = {new typeT/S(…), new typeT/S(…), …};

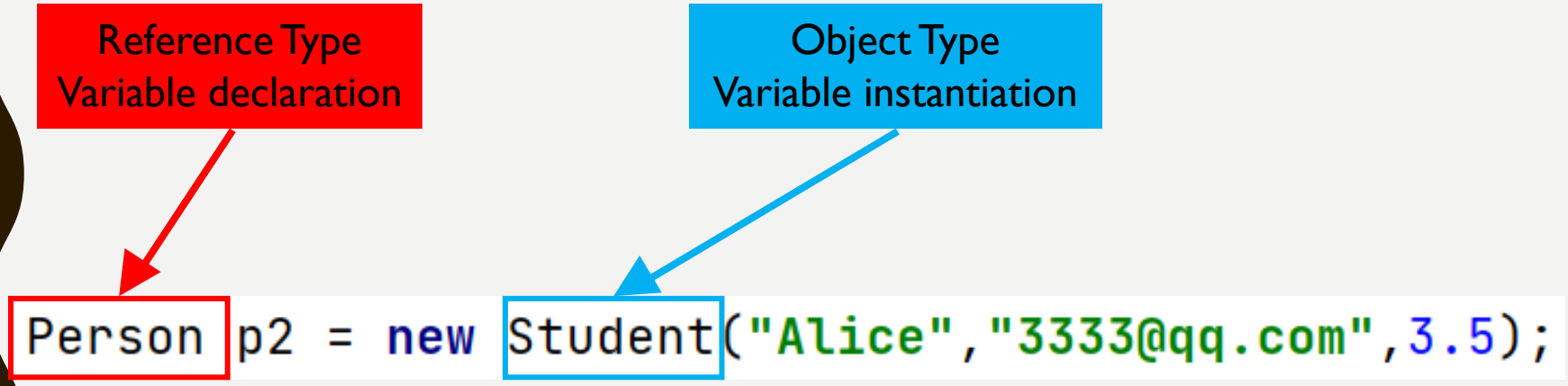
      name[0] = new typeT/S(…);
    - **Array List:** ArrayList<type T> name = new ArrayList<type T/S>();

      name.add(new typeT/S(…));

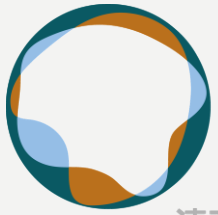


COMPILE-TIME

At the time a program gets **compiled**, methods in or inherited by the declare/reference type determine the correctness of a non static method call. (not object type)



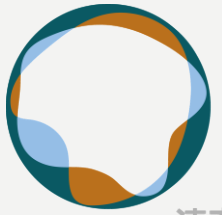
See Code



RUN-TIME

At the time a program **runs**, the methods in the actual object type gets executed. If the method doesn't exist there, Java looks to the superclass for the method.

See Code



COMPILE-TIME VS RUNTIME

- An error is a **compile-time error** if it happens when the program compiles.
  - All method overloading errors are compile-time errors.
  - missing semicolons, curly braces.

- An error is a **runtime error** if it happens when the program runs.
  - Casting too far down, sideways are run-time errors.
  - divide by zero, out of bounds index errors

A runtime error compiles without errors.

THE COSMIC SUPERCLASS OBJECT

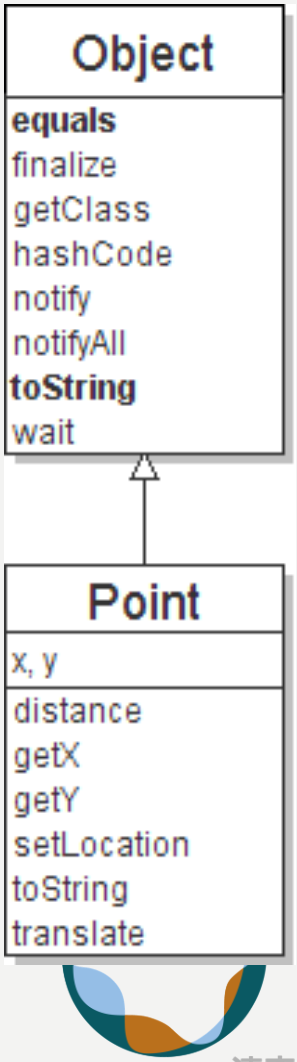
All types of objects have a superclass named `Object`.

- Every class implicitly extends `Object`

The `Object` class defines several methods:

- `public String toString()`
  Returns a text representation of the object, often so that it can be printed. We have seen this in Unit 5.

- `public boolean equals(Object other)`
  Compare the object to any other for equality. Returns `true` if the objects have equal state.



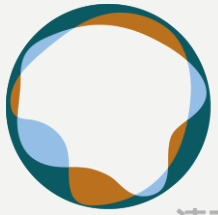
OBJECT VARIABLES

You can store any object in a variable of type `Object`.

```
Object o1 = new Point(5, -3);
Object o2 = "hello there";
Object o3 = new Scanner(System.in);
```

An `Object` variable only knows how to do general things.

```
String s = o1.toString(); // ok(memory address)
int len = o2.length();    // compile-time error
String line = o3.nextLine(); // compile-time error
```

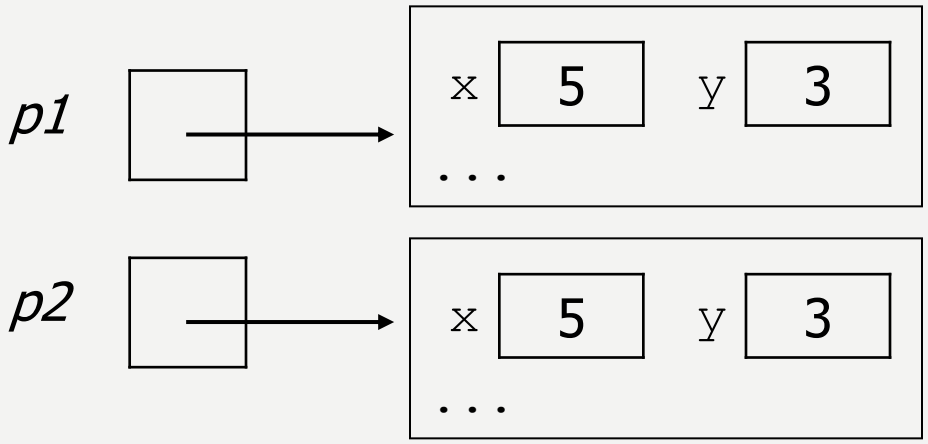


RECALL: COMPARING OBJECTS

The == operator does not work well with objects.

- == compares references to objects, not their state. It only produces `true` when you compare an object to itself.

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
if (p1 == p2) {    // false
    System.out.println("equal");
}
```



THE EQUALS METHOD

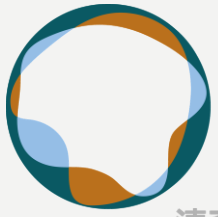
The `equals` method compares the state of objects in String class.

```
if (str1.equals(str2)) {
    System.out.println("the strings are equal");
}
```

But if you write a class, its `equals` method behaves like `==`

```
if (p1.equals(p2)) {    // false :-(
    System.out.println("equal");
}
```

- This is the behavior we inherit from class `Object`.
- Java doesn't understand how to compare `Point`s by default.



EQUALS METHOD

We can change this behavior by writing an `equals` method that **overrides** the one inherited from `Object`.

- **Note the method header including the parameter Object o below.**
- The method should compare the state of the two objects and return `true` if they have the same x/y position.

```
public boolean equals(Object o) {
        Point other = (Point) o;
    return (x == other.x && y == other.y)
}
```

AN IMPLEMENTATION OF POINT

Here's the Point class with both `toString` and `equals` overriden.

```
public class Point {
    private int x;
    private int y;
  public Point(int newX, int newY){
      x = newX;
      y = newY;
  }
    public boolean equals(Object o) {
          Point other = (Point) o;
        return (x == other.x && y == other.y);
    }
    public String toString(){
          return "(" + x + ", " + y + ")";
    }

}
```

MAIN

```
public class Main {
  public static void main(String[] args){
    Point x = new Point(2, -5);
    Point y = new Point(2, -5);
    Point z = new Point(3, 8);
    Point w = z;
    System.out.println(x == y); // false
    System.out.println(x.equals(y)); // true
    System.out.println(z == w); // true
    System.out.println(x.equals(w)); // false
    System.out.println(x); //(2, -5)
    // call toString() implicitly
    System.out.println(z.toString()); //(3, 8)
  }
```

x and y are two different objects but mathematically equivalent. Overriding equals allows us to easily recognize that certain objects are equivalent.