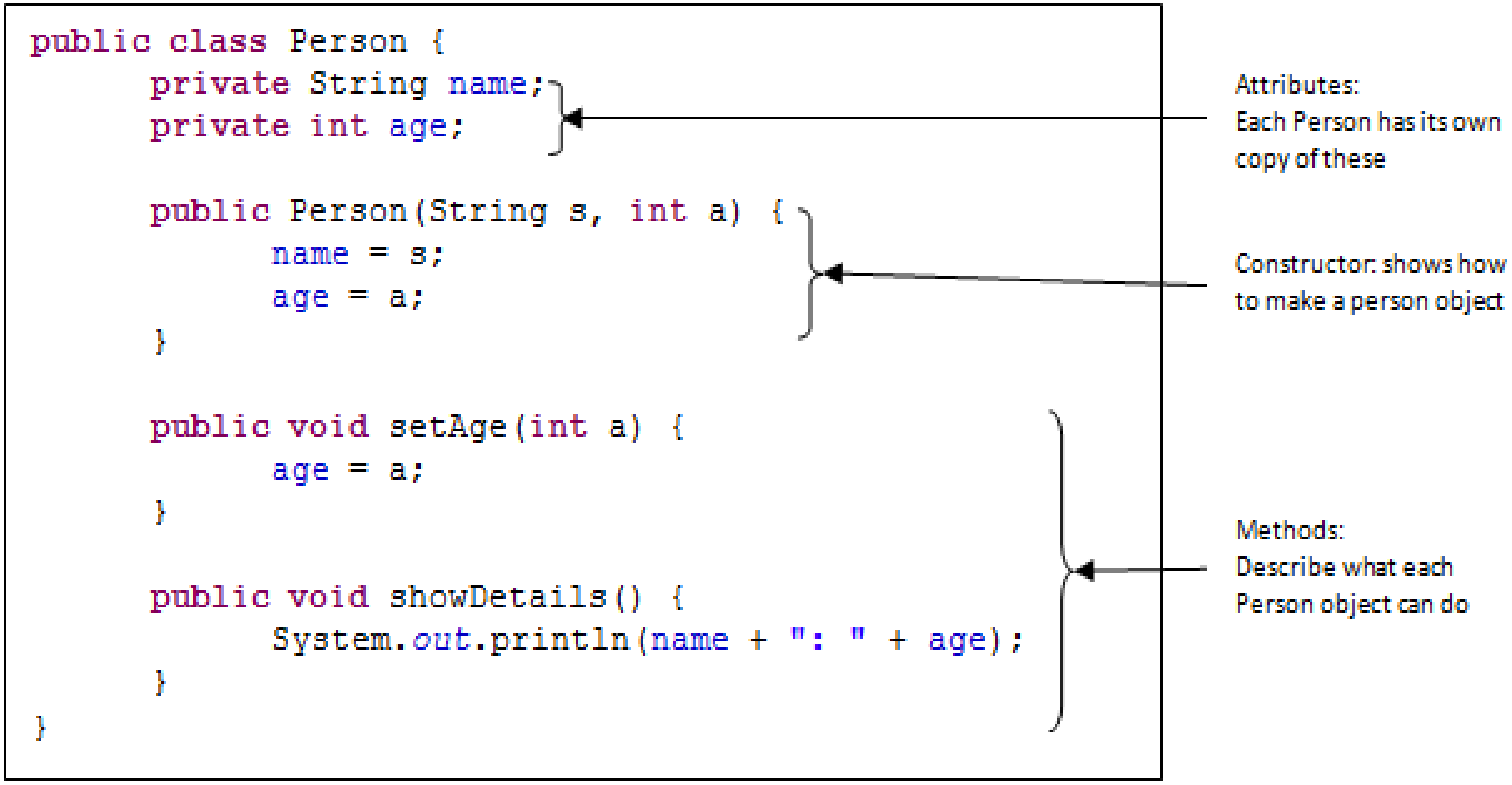
Objects and Classes

Review – Object Orientation

- Object orientation focuses on data
  - functionality is associated with the data elements
- Objects have three primary properties
  - attributes(variables)
  - identity(where it lives / name)
  - behaviour(methods)
- Objects of a given type belong to a class
  - Template for building objects
- Classes may be related by inheritance
  - define one class in terms of another
  - brings many advantages

A Simple Class



```java
public class Person {
	private String name;
	private int age;

	public Person(String s, int a) {
		name = s;
		age = a;
	}

	public void setAge(int a) {
		age = a;
	}

	public void showDetails() {
		System.out.println(name + ": " + age);
	}
}
```

Attributes:
Each Person has its own copy of these

Constructor: shows how to make a person object

Methods:
Describe what each Person object can do

Using the Class

```java
public class PersonTest{
      public static void main(String[] args){
            Person nilesh;
            Person rishabh;

            nilesh=new Person("Nilesh",22);
            nilesh.showDetails();

            rishabh=new Person("Rishabh", 21);
            rishabh.showDetails();

            nilesh.setAge(23);
            nilesh.showDetails();
      }
}
```

declaration

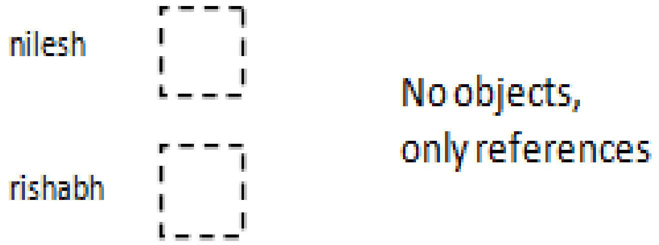
initialization

```
$ java PersonTest
Nilesh: 22
Rishabh: 21
Nilesh: 23
```

Declaring and Creating Objects

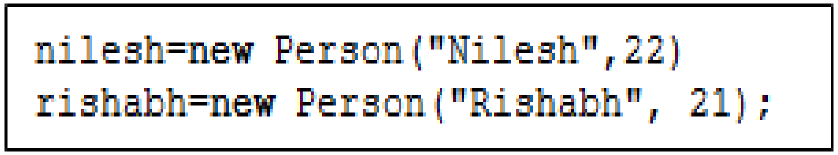
- Object declarations are references
  - they must be associated with an object before use

```
Person nilesh;
Person rishabh;
```

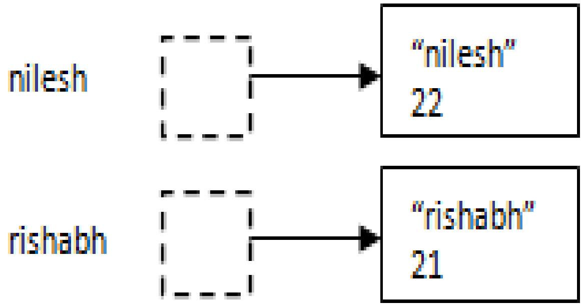


Declaring and Creating Objects

- Create Objects using the `new` operator

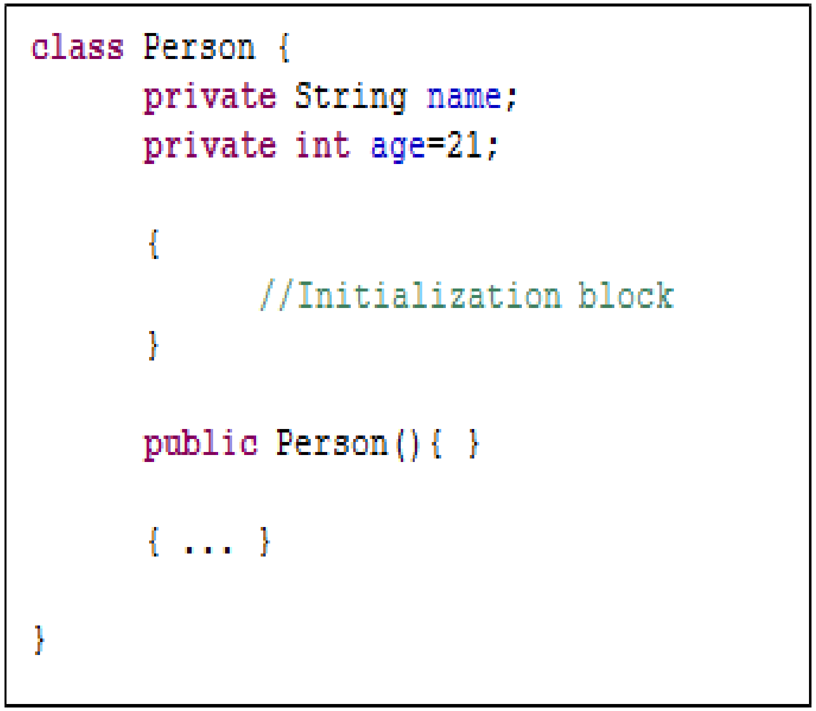


```
nilesh=new Person("Nilesh",22)
rishabh=new Person("Rishabh", 21);
```



Object Initialisation

- Objects created with `new` operator
- Attributes set to default values
    - 0 for numeric/char primitive types
    - false for boolean
    - null for objects
- Class specific initial values then set
- Initialization block executed
- Constructor called for more detailed initialisation



```
class Person {
        private String name;
        private int age=21;

        {
                //Initialization block
        }

        public Person(){ }

        { ... }

}
```

The Constructor

- Pseudo-method to initialise newly created object
  - same name as class, no return type
- May be overloaded
  - correct constructor called according to argument list used with `new`
- Default “no-arg” constructor provided
  - unless class contains other constructors

The Constructor

```
class Person {
      private String name;
      private int age=21;

      public Person(String n){
            name=n;
      }

      public Person(String s, int a) {
            name = s;
            age = a;
      }
}
```

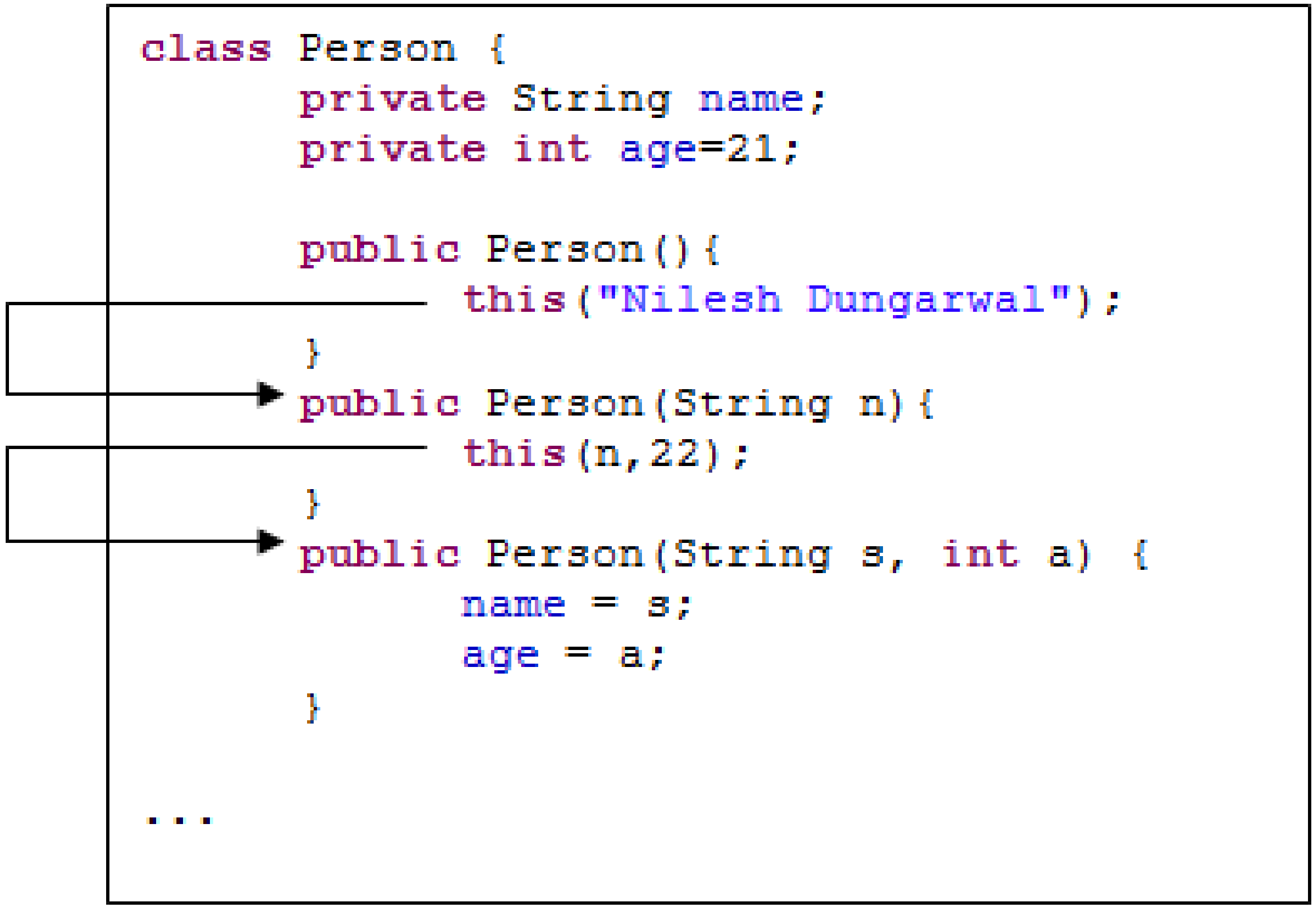
```
nilesh=new Person("Nilesh",22);

rishabh=new Person("Rishabh", 21);

yash=new Person();
```

Not Allowed !!!
No Constructor with matching signature

Constructor Cascades

- Useful for building objects that use default values for attributes
  - localises processing
  - avoid code duplication
- Use `this()` to invoke other constructors
  - must be first statement in constructor

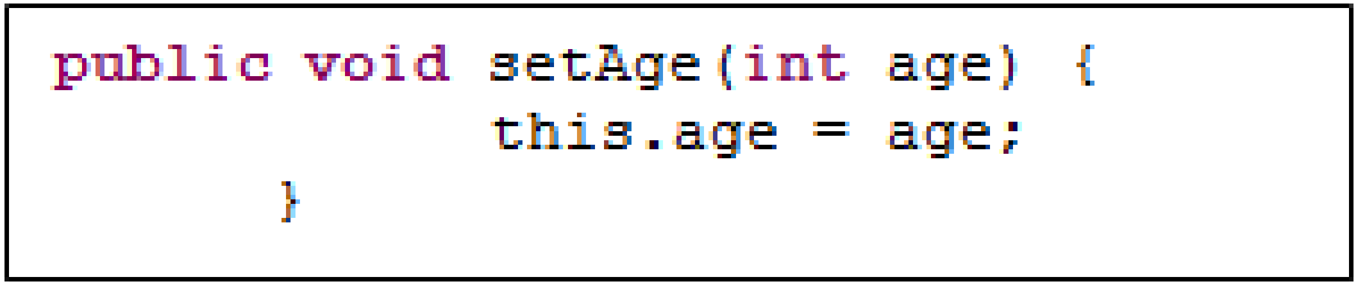
Constructor Cascades



```
class Person {
        private String name;
        private int age=21;

        public Person(){
              this("Nilesh Dungarwal");
        }
        public Person(String n){
              this(n,22);
        }
        public Person(String s, int a) {
              name = s;
              age = a;
        }

...
```

The Current Object

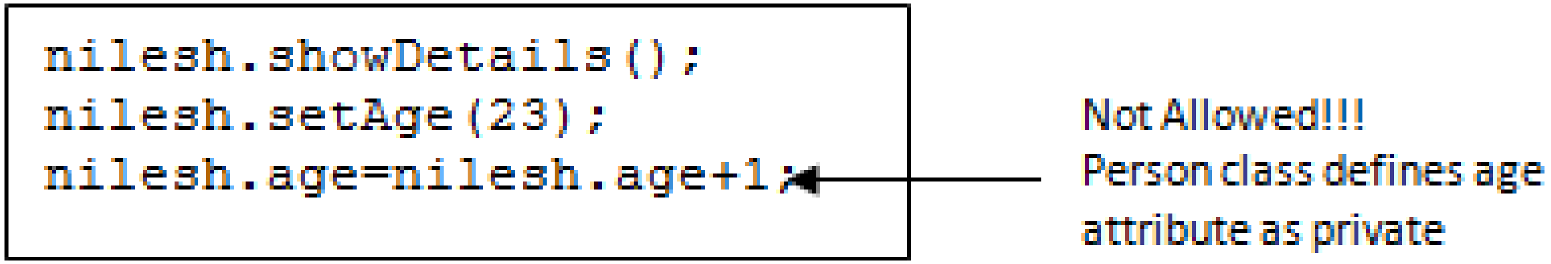
- `this` reference
- Available in constructor and methods
    - not static methods (like `main()`)
- Often used to avoid possible ambiguity



```
public void setAge(int age) {
                this.age = age;
        }
```

Working with Objects

- Use the `'.'` operator to access methods and attributes
  - subject to visibility rules defined in clas



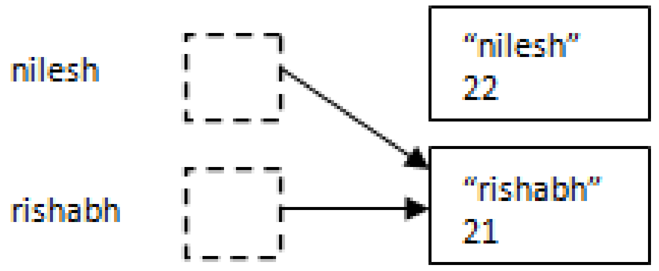
```
nilesh.showDetails();
nilesh.setAge(23);
nilesh.age=nilesh.age+1;
```

Not Allowed!!!
Person class defines age attribute as private

Working with Objects

- Beware assignment
  - assigns reference, not object!!

```
nilesh = rishabh;
nilesh.setAge(23);
rishabh.showDetails();
```



Copying Objects

- Use copy constructor to create copy of an object
  - constructor that uses existing object for initialisation
- Alternative is `clone()` method
  - implemented within class
  - often uses copy constructor

Copying Objects

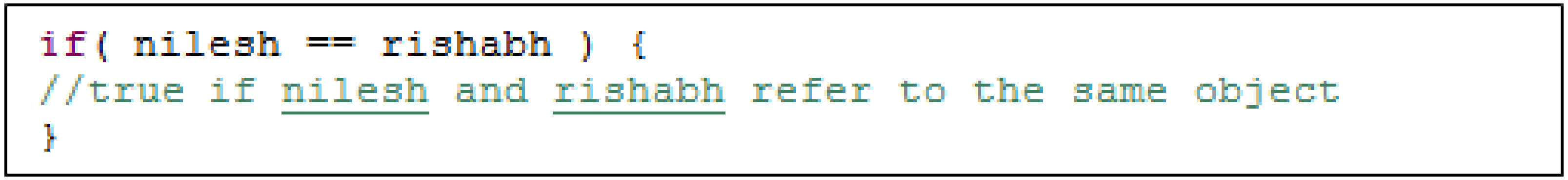
```
public class Person{
...
      public Person ( Person p ){
      this.name = p.name;
      this.age = p.age;
      }
...
}
```

```
Person nilesh = new Person(“nilesh”,22);
Person newNilesh = new Person(nilesh);
```

```
Person original = new Person(“nilesh”,22);
Person cloned = (Nilesh)original.clone()
```

Object Equality

- Care required when comparing objects
- Normal == operator compares object identity
  - do the references point to the same object?



```
if( nilesh == rishabh ) {
//true if nilesh and rishabh refer to the same object
}
```

Object Equality

- Class implementer provides `equals()` method
  - determines whether objects are to be considered equal
  - conditions for equality are defined by class designer

```
if( nilesh.equals(rishabh) ) {
//true if nilesh and rishabh refer to the objects
//considered to be equal
}
```

```
public boolean equals(Object obj)  {
        return (this == obj);
}
```

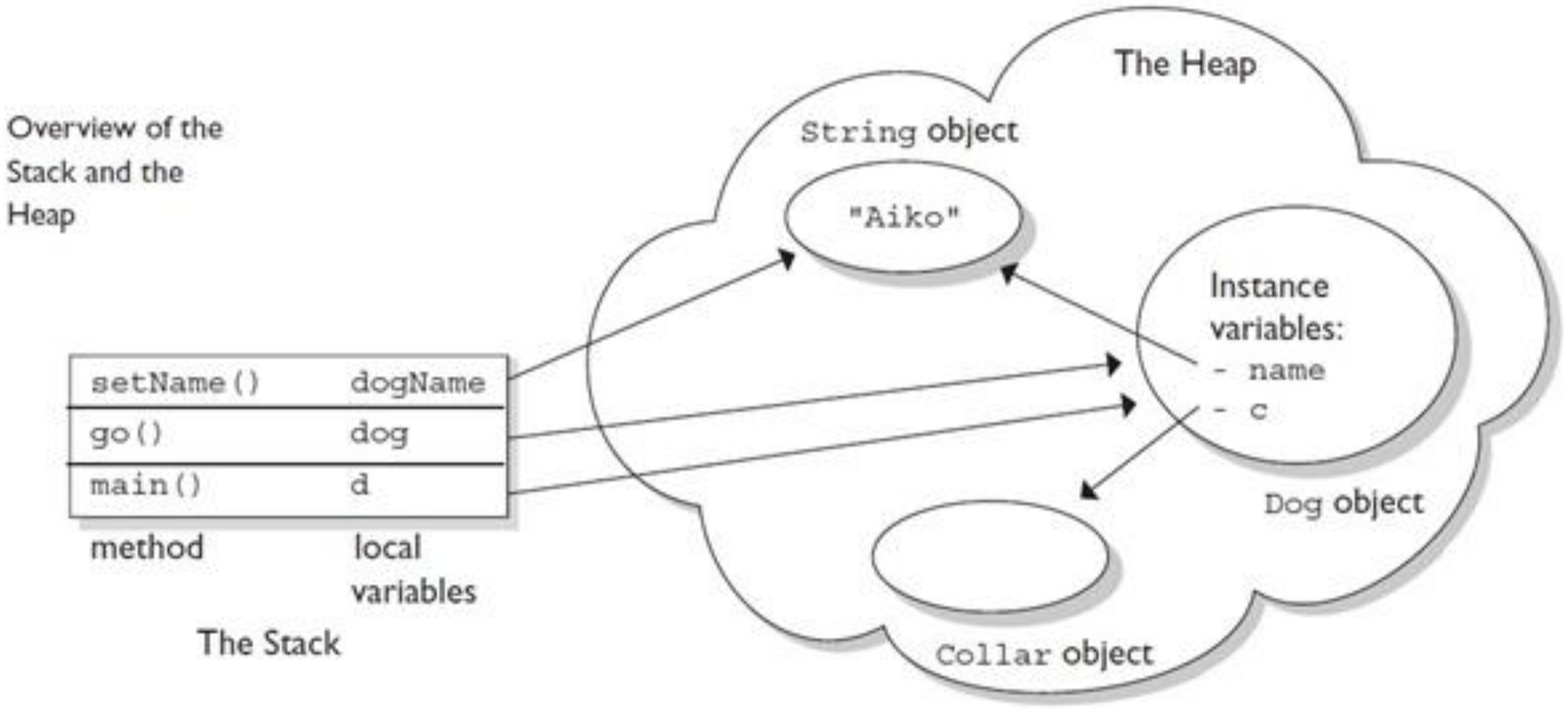
Stack and Heap

- The various pieces of Java Program live in one of two places in memory
- Three types of things
  - instance variables live on heap
  - objects live on heap
  - Local variables live on stack

Stack and Heap

```java
class Collar { }

class Dog {
Collar c; // instance variable
String name; // instance variable

public static void main(String [] args) {

 Dog d; // local variable: d
d = new Dog();
d.go(d);
}
void go(Dog dog) { // local variable: dog
c = new Collar();
dog.setName("Aiko");
}
void setName(String dogName) { // local var: dogName
name = dogName;
// do more stuff
}
```

Stack and Heap



Garbage Collector

- Java provides automatic memory management
- Under control of JVM
- No guarantees when garbage collector will run
- Explicit request for garbage collection using `System.gc()` but there are no guarantees

How does Garbage Collector works?

- Specification does not guarantee any Java implementation. But you might hear
  - Mark and sweep algorithm
  - Reference counting
- It may be yes may be no

Eligibility for Garbage Collection

- Every java program has one or many threads
- Threads can be alive or dead
- An object is eligible for garbage collection when no live thread can access it
- Garbage collector does some unknown magical operations

Explicitly make objects eligible for garbage collection

- Nulling a reference

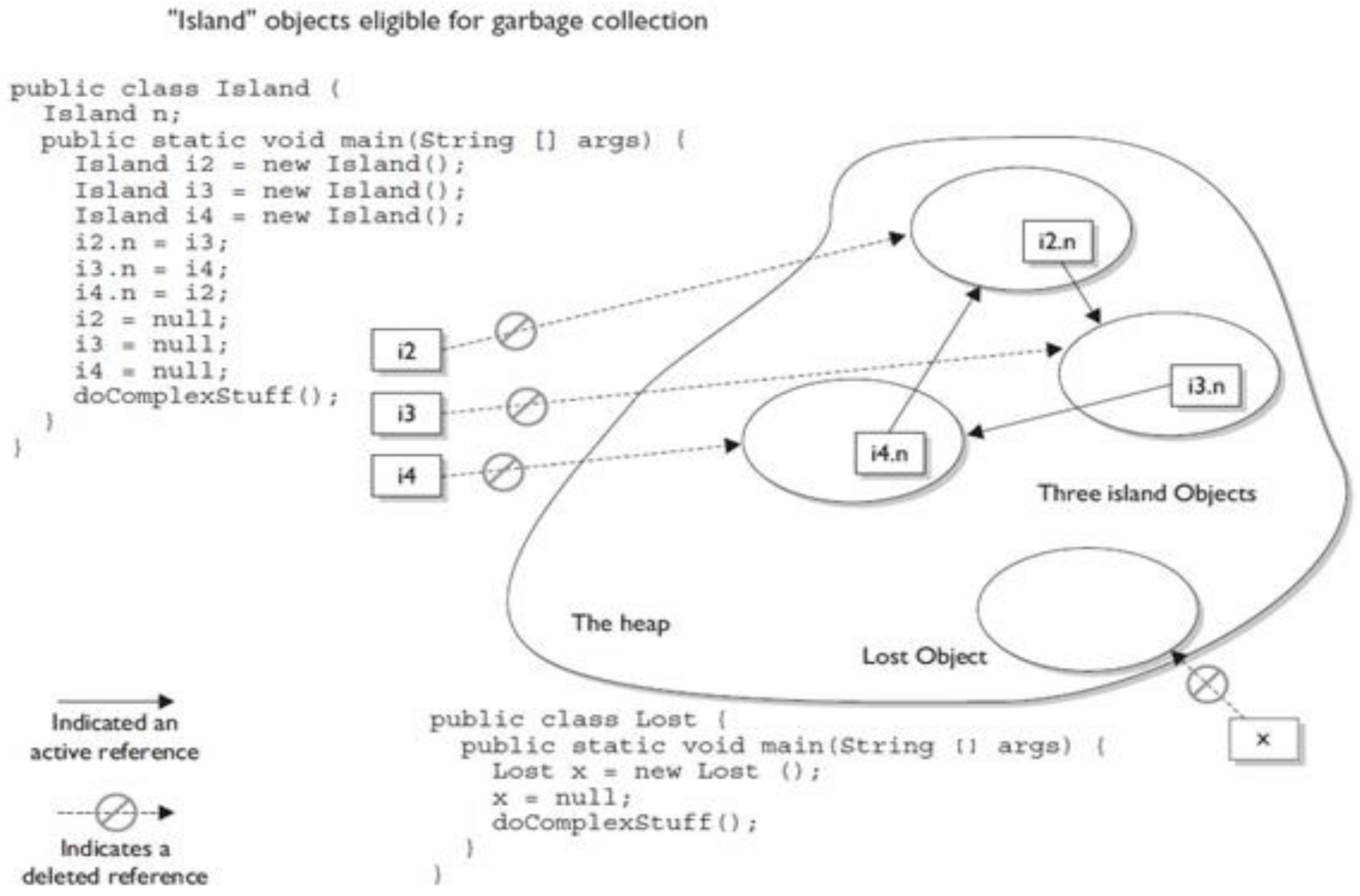
```
Person nilesh = new Person();
nilesh = null //Eligible for Garbage Collection
```

- Reassigning a reference variable

```
Person nilesh = new Person();
Person rishabh = new Person();

nilesh = rishabh //nilesh eligible for Garbage Collection
```

- Isolating a reference

Islands of Isolation


"Island" objects eligible for garbage collection
i2
i3
i4
i2.n
i3.n
i4.n
Three island Objects
The heap
Lost Object
x
Indicated an active reference
Indicates a deleted reference

```
public class Island {
  Island n;
  public static void main(String [] args) {
    Island i2 = new Island();
    Island i3 = new Island();
    Island i4 = new Island();
    i2.n = i3;
    i3.n = i4;
    i4.n = i2;
    i2 = null;
    i3 = null;
    i4 = null;
    doComplexStuff();
  }
}
```

```
public class Lost {
  public static void main(String [] args) {
    Lost x = new Lost ();
    x = null;
    doComplexStuff();
  }
}
```

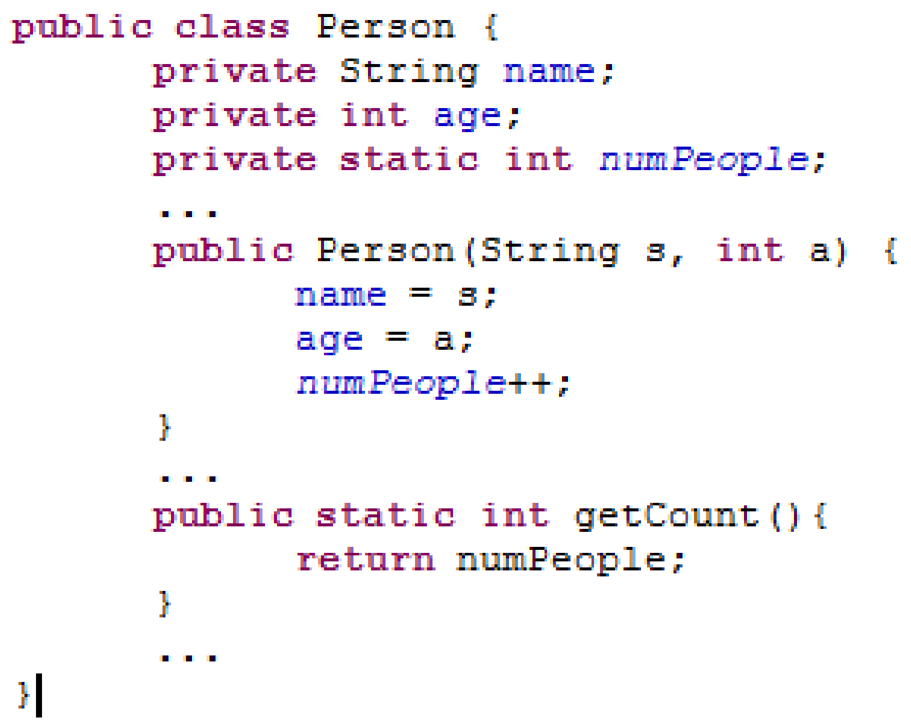
Object Lifetime

- Object “lives” as long as someone reference it
- Space reclaimed for the heap by the garbage collector
- `finalize()` method called when garbage collector reclaims object
  - if defined in object’s class
  - designed to contain clean-up functionality
- Reclaim operation is non-deterministic
  - may not even happen
  - `finalize()` may never be called

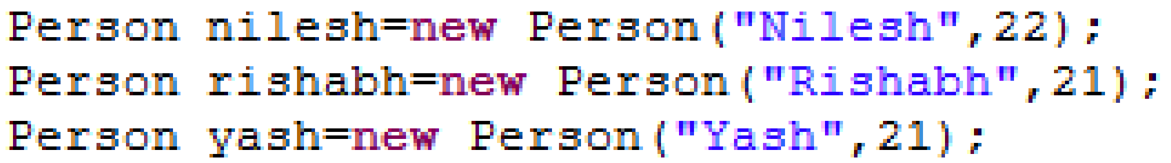
Class Data & Methods

- Methods and attributes may be declared as `static`
  - relate to class rather than objects
  - do not need objects to be created
  - qualified by class name rather than object reference
  - no `this` reference
  - cannot access object methods or data without object reference
- Static methods are bound to class

Class Data & Methods



```
public class Person {
        private String name;
        private int age;
        private static int numPeople;
        ...
        public Person(String s, int a) {
                name = s;
                age = a;
                numPeople++;
        }
        ...
        public static int getCount(){
                return numPeople;
        }
        ...
}
```



```
Person nilesh=new Person("Nilesh",22);
Person rishabh=new Person("Rishabh",21);
Person yash=new Person("Yash",21);
```

Questions

- What is Classes and Objects?
- How is Object initialised?
- What is Constructor?
- What is `this` reference?
- What is the life of an Object?
- What happens when an Object is destroyed?
- What is the difference between == and equals() method
- What is Garbage Collector and how does it works?
- What is Stack and Heap for memory allocation in Java?

Contact Info

- trainers@finaldesk.com
- rishabh@finaldesk.com
- nilesh@finaldesk.com
- jignesh@finaldesk.com
- yash@finaldesk.com
- anand@finaldesk.com