

MASTER SLIDES & REFERENCE

TYPES OF RELATIONSHIPS

- **Is-a relationship** is one in which data members of one class is obtained into another class through the concept of inheritance.
- **Has-a relationship** is one in which an object of one class is created as a data member in another class.
- **Uses-a relationship** is one in which a method of one class is using an object of another class.

```
class C1
```

```
{  
.....  
.....  
}  
  
class C2 extends C1  


```
{
.....
.....
}
```


```

Is-A Relation

```
class C1
```

```
{  
.....  
.....  
}  
  
class C2  


```
{
.....
.....
}
```


```

Has-A Relation

```
C1 obj = new C1();  
.....  
.....
```

```
class C1  
{  
.....  
.....  
}  
  
class C2  


```
{
.....
.....
void disp()
{
.....
.....
C1 obj=new C1();
.....
.....
}
}
```


```

Uses-A
Relation

Note 1: The default relationship in java is Is-A because for each and every class in java there exist an implicit predefined super class is `java.lang.Object`.

Note 2: The universal example for Has-A relationship is `System.out` (in `System.out` statement, `out` is an object of `printStream` class created as static data member in another system class and `printStream` class is known as Has-A relationship).

Note 3: Every execution logic method (`main()`) of execution logic is making use of an object of business logic class and business logic class is known as Uses-A relationship.

VARARGS

- You can use a construct called *varargs* to pass an arbitrary number of values to a method.
- You use varargs when you don't know how many of a particular type of argument will be passed to the method. It's a shortcut to creating an array manually
- **Important Note:** The parameter(s) passed in this way is always an array - even if there's just one. Make sure you treat it that way in the method body.
- **Important Note 2:** The parameter that gets the ... must be the last in the method signature. So, myMethod(int i, String... strings) is okay, but myMethod(String... strings, int i) is not okay.

JAVA KEYWORDS

Java Language Keywords

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords `const` and `goto` are reserved, even though they are not currently used. `true`, `false`, and `null` might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

<code>abstract</code>	<code>continue</code>	<code>for</code>	<code>new</code>	<code>switch</code>
<code>assert^{**}</code>	<code>default</code>	<code>goto[*]</code>	<code>package</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>private</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>protected</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>public</code>	<code>throws</code>
<code>case</code>	<code>enum^{****}</code>	<code>instanceof</code>	<code>return</code>	<code>transient</code>
<code>catch</code>	<code>extends</code>	<code>int</code>	<code>short</code>	<code>try</code>
<code>char</code>	<code>final</code>	<code>interface</code>	<code>static</code>	<code>void</code>
<code>class</code>	<code>finally</code>	<code>long</code>	<code>strictfp^{**}</code>	<code>volatile</code>
<code>const[*]</code>	<code>float</code>	<code>native</code>	<code>super</code>	<code>while</code>

* not used

** added in 1.2

*** added in 1.4

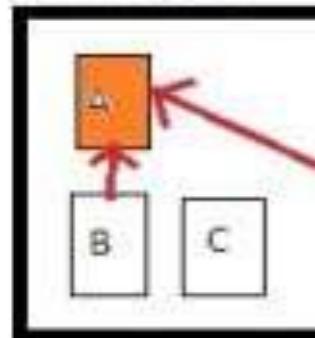
**** added in 5.0

DATA TYPE SIZE

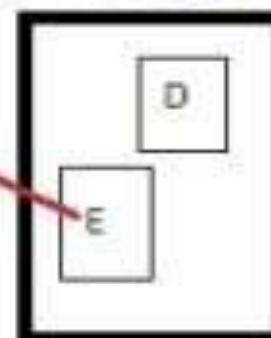
Data Type	Bits	Range
byte	8	-2^7 to 2^7-1 (-128 to 127)
short	16	-2^{15} to $2^{15}-1$ (-32,768 to 32,767)
int	32	-2^{31} to $2^{31}-1$ (-2,147,483,648 to 2,147,483,647)
long	64	-2^{63} to $2^{63}-1$ (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
float	32	1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative) (single-precision 32-bit IEEE 754 floating point)
double	64	4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative) (double-precision 64-bit IEEE 754 floating point)
char	16 (unsigned)	0 to 65,535

ACCESS MODIFIERS

Package 1

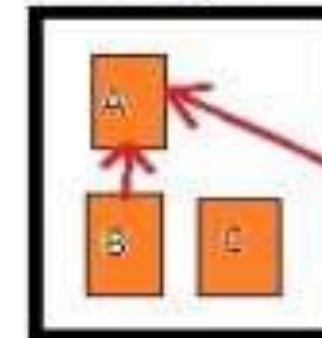


Package 2

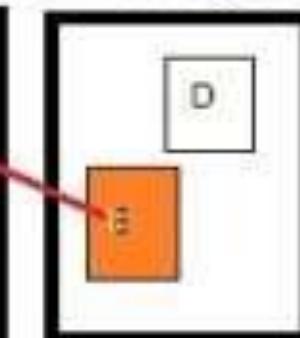


PRIVATE

Package 1

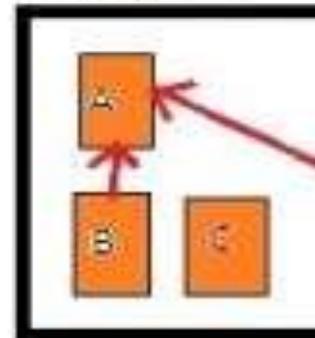


Package 2

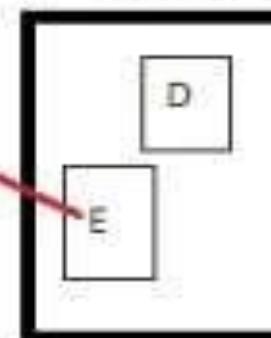


PROTECTED

Package 1

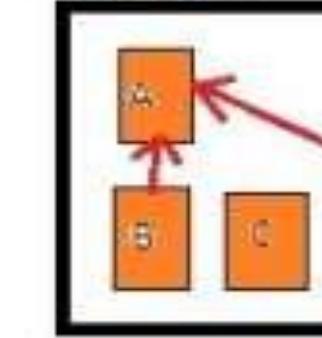


Package 2

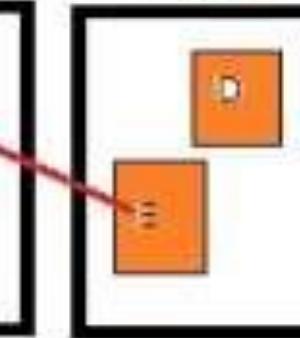


DEFAULT

Package 1



Package 2



PUBLIC

ACCESS MODIFIERS

Visibility	Public	Protected	Default	Private
From the same class	Yes	Yes	Yes	Yes
From any class in the same package	Yes	Yes	Yes	No
From a subclass in the same package	Yes	Yes	Yes	No
From a subclass outside the same package	Yes	Yes, through inheritance	No	No
From any non-subclass class outside the package	Yes	No	No	No

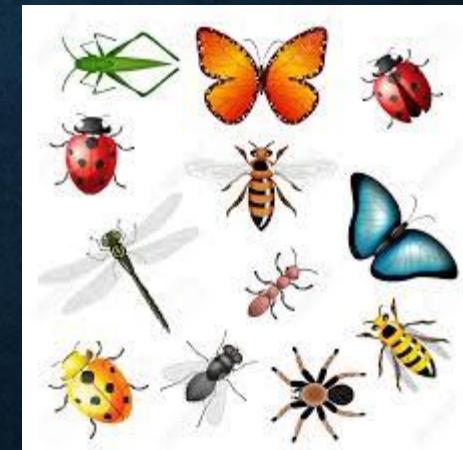
MODIFIERS

Modifiers-Elements Matrix in Java

element	Data field	Method	Constructor	Class		Interface	
modifier				top level (outer)	nested (inner)	top level (outer)	nested (inner)
abstract	no	yes	no	yes	yes	yes	yes
final	yes	yes	no	yes	yes	no	no
native	no	yes	no	no	no	no	no
private	yes	yes	yes	no	yes	no	yes
protected	yes	yes	yes	no	yes	no	yes
public	yes	yes	yes	yes	yes	yes	yes
static	yes	yes	no	no	yes	no	yes
synchronized	no	yes	no	no	no	no	no
transient	yes	no	no	no	no	no	no
volatile	yes	no	no	no	no	no	no
strictfp	no	yes	no	yes	yes	yes	yes

THE JAVA COLLECTION FRAMEWORK (JCF)

- A *collection* — sometimes called a *container* — is simply an object that groups multiple elements into a single unit.
- Collections are used to store, retrieve, manipulate, and communicate aggregate data.
- The collection framework gives you lists, sets, and maps to satisfy most of your coding needs.



JAVA COLLECTION FRAMEWORK (JCF)

- A *collections framework* is a unified architecture for representing and manipulating collections.
- All collections frameworks contain the following:
 - **Interfaces:** These are abstract data types that represent collections. Interfaces allow collections to be manipulated independently of the details of their representation. In object-oriented languages, interfaces generally form a hierarchy.
 - **Implementations:** These are the concrete implementations of the collection interfaces. In essence, they are reusable data structures.
 - **Algorithms:** These are the methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. The algorithms are said to be *polymorphic*: that is, the same method can be used on many different implementations of the appropriate collection interface. In essence, algorithms are reusable functionality.

OPERATIONS USING COLLECTIONS

- There are a few basic operations you'll normally use with collections:
 - Add objects to the collection.
 - Remove objects from the collection.
 - Find out if an object (or group of objects) is in the collection.
 - Retrieve an object from the collection (without removing it).
 - Iterate through the collection, looking at each element (object) one after another.
 - The collection API begins with a group of interfaces, but also gives you a truckload of concrete classes.

3 BASIC FLAVORS

- Lists: *Lists* of things (classes that implement List)
- Sets: *Unique* things (classes that implement Set)
- Maps: Things with a *unique ID* (classes that implement Map)



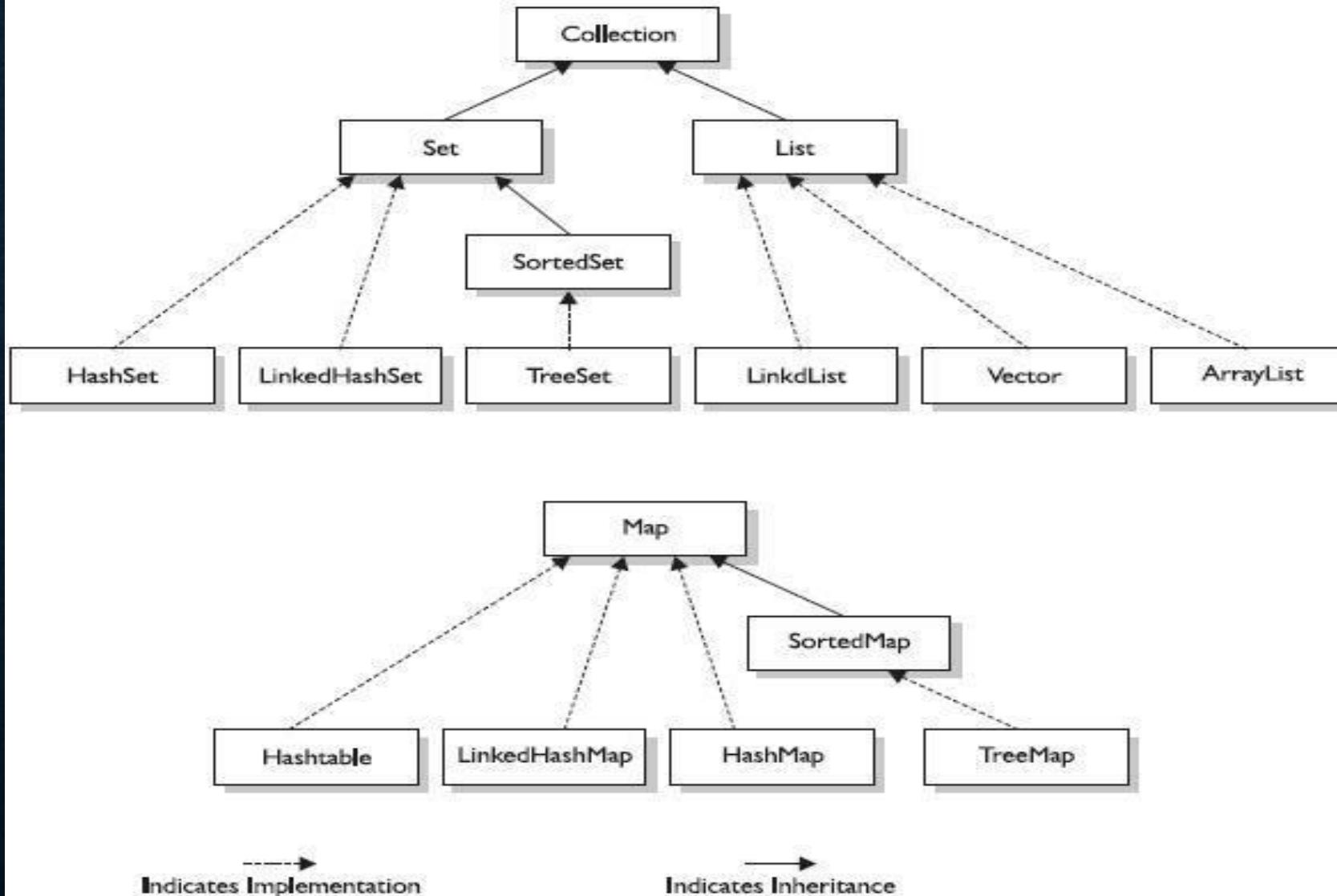
4 SUB FLAVORS

- Sorted
- Unsorted
- Ordered
- Unordered



THE COLLECTION API

The collections class and interface hierarchy



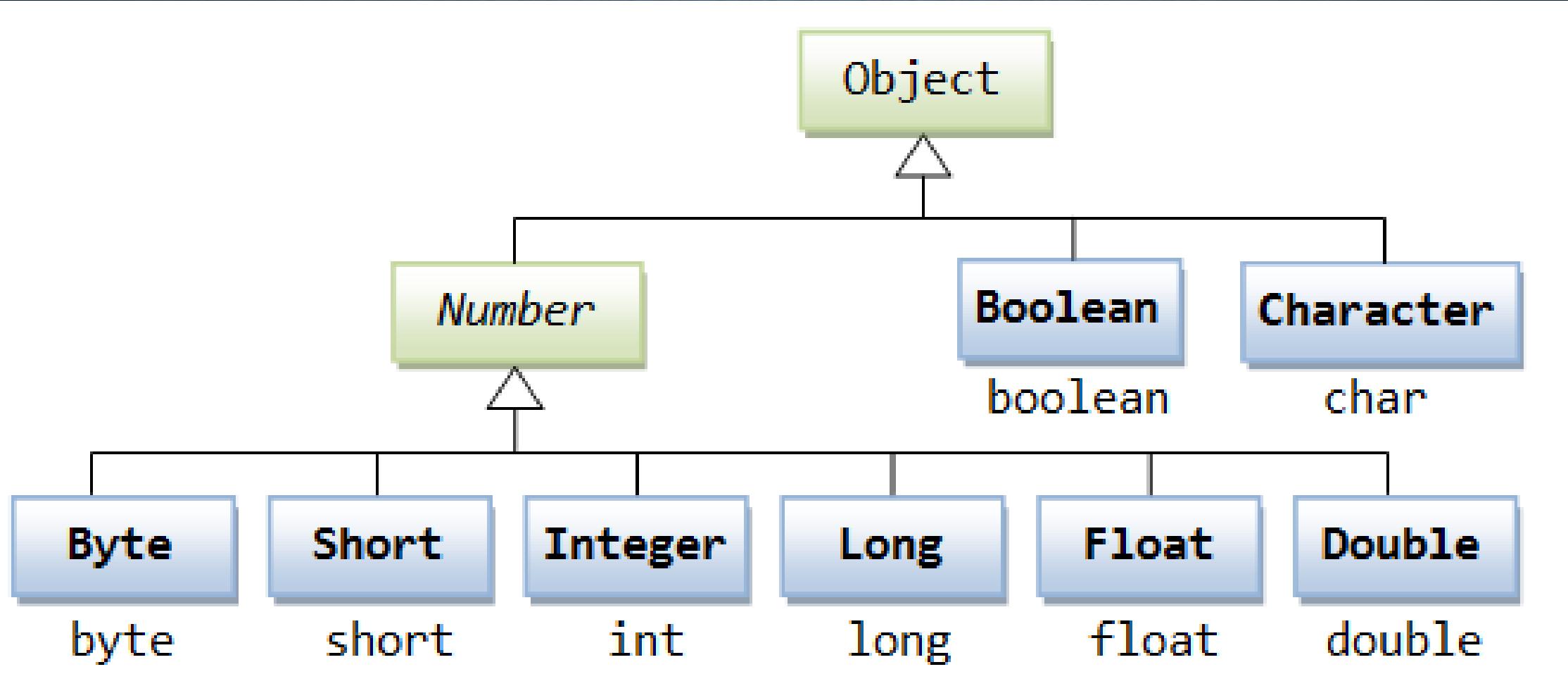
THE COLLECTION API

- The core interfaces you need to know are the following six:
 - Collection
 - Set
 - SortedSet
 - List
 - Map
 - SortedMap
- Map implementation:
 - HashMap, Hashtable, TreeMap and LinkedHashMap.
- Set implementation:
 - HashSet, LinkedHashSet and TreeSet.
- List implementation:
 - ArrayList, Vector and LinkedList.

COLLECTIONS CHEAT SHEET

	ArrayList	Vector	LinkedList	HashMap	LinkedHashMap	HashTable	TreeMap	HashSet	LinkedHashSet	TreeSet
Allows Null?	Yes	Yes	Yes	Yes (But One Key & Multiple Values)	Yes (But One Key & Multiple Values)	No	Yes (But Zero Key & Multiple Values)	Yes	Yes	No
Allows Duplicates?	Yes	Yes	Yes	No	No	No	No	No	No	No
Retrieves Sorted Results?	No	No	No	No	No	No	Yes	No	No	Yes
Retrieves Same as Insertion Order?	Yes	Yes	Yes	No	Yes	No	No	No	Yes	No
Synchronized?	No	Yes	No	No	No	Yes	No	No	No	No

WRAPPER CLASSES



WRAPPER CLASSES

TABLE 3-2

Wrapper Classes and Their Constructor Arguments

Primitive	Wrapper Class	Constructor Arguments
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
double	Double	double or String
float	Float	float, double, or String
int	Integer	int or String
long	Long	long or String
short	Short	short or String

DATA TYPE CONVERSIONS

- BOXING & UNBOXING
- Primitive Type >> Reference Type
- The process of encapsulating a value within an object is called *boxing*.
 - `Integer iOb = new Integer(100);`
- The process of extracting a value from a type wrapper is called *unboxing*.
- *Prior to JDK 5 all boxing and unboxing took place manually.*
 - `int i = iOb.intValue();`



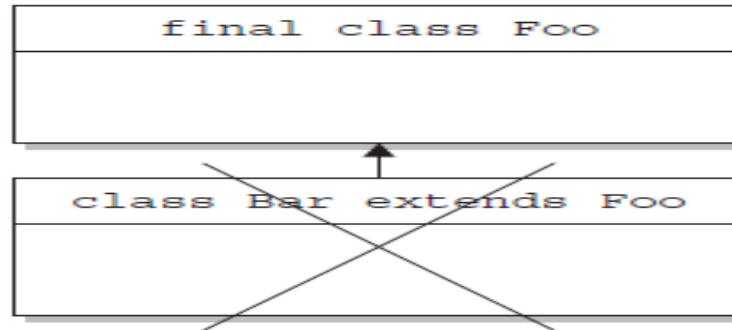
JDK 5 AND LATER ☺



- Autoboxing is the process by which a primitive type is automatically encapsulated (boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.
- Auto-unboxing is the process by which the value of a boxed object is automatically extracted (unboxed) from a type wrapper when its value is needed. There is no need to call a method such as **intValue()** or **doubleValue()**.

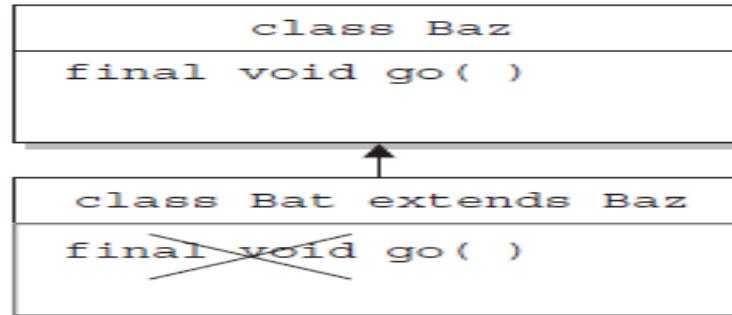
THE FINAL KEYWORD

**final
class**



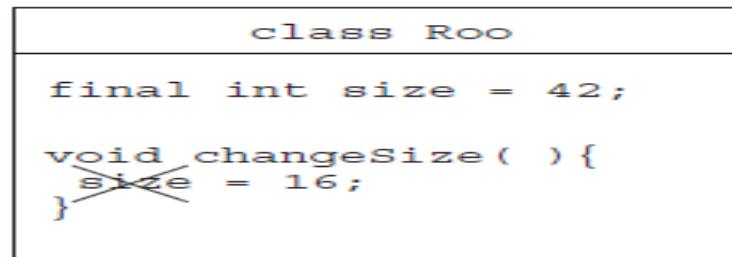
**final class
cannot be
subclassed**

**final
method**



**final method
cannot be
overridden by
a subclass**

**final
variable**



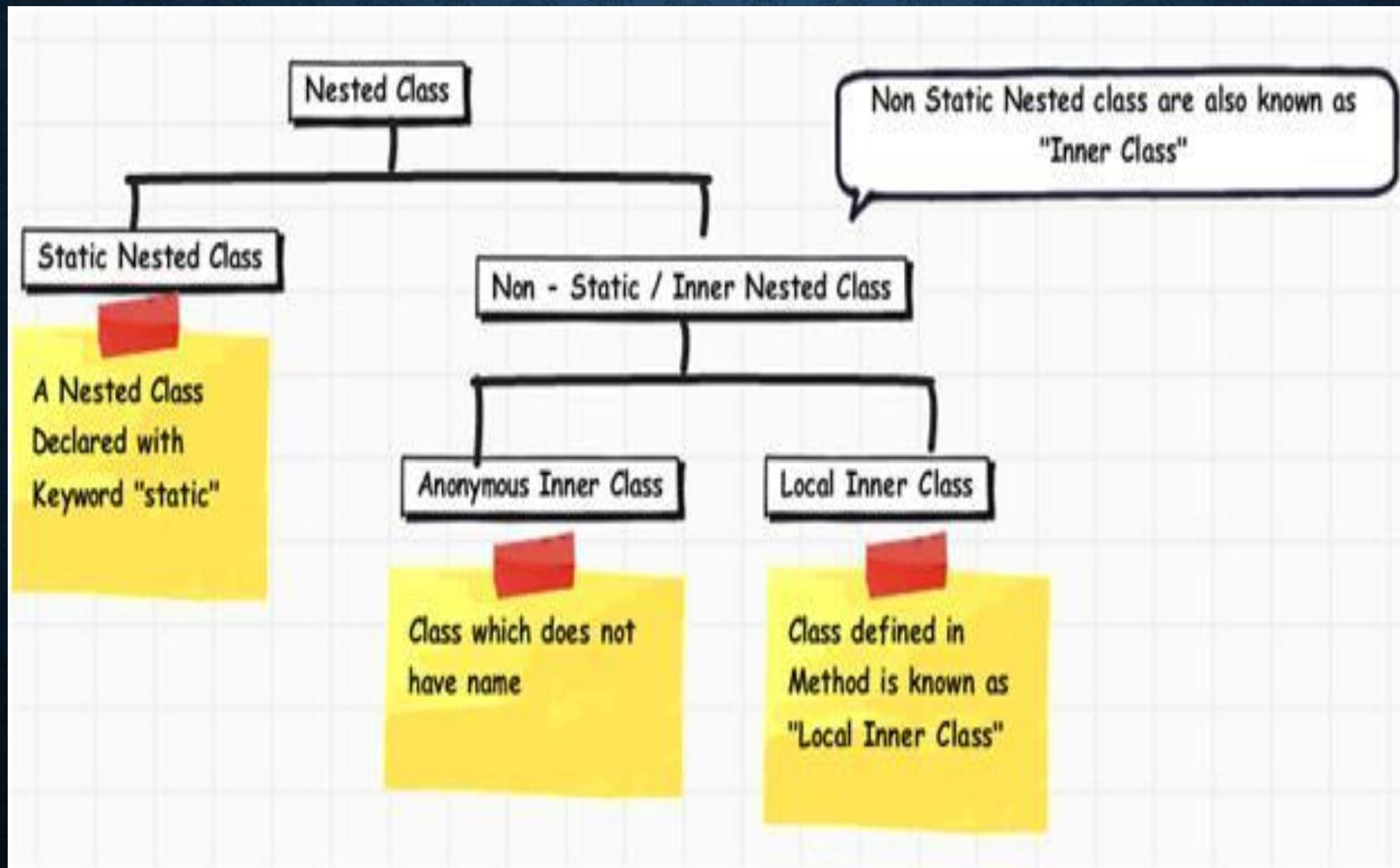
final variable cannot be
assigned a new value, once
the initial method is made
(the initial assignment of a
value must happen before
the constructor completes).

THE JAVA.LANG.OBJECT

Method Summary

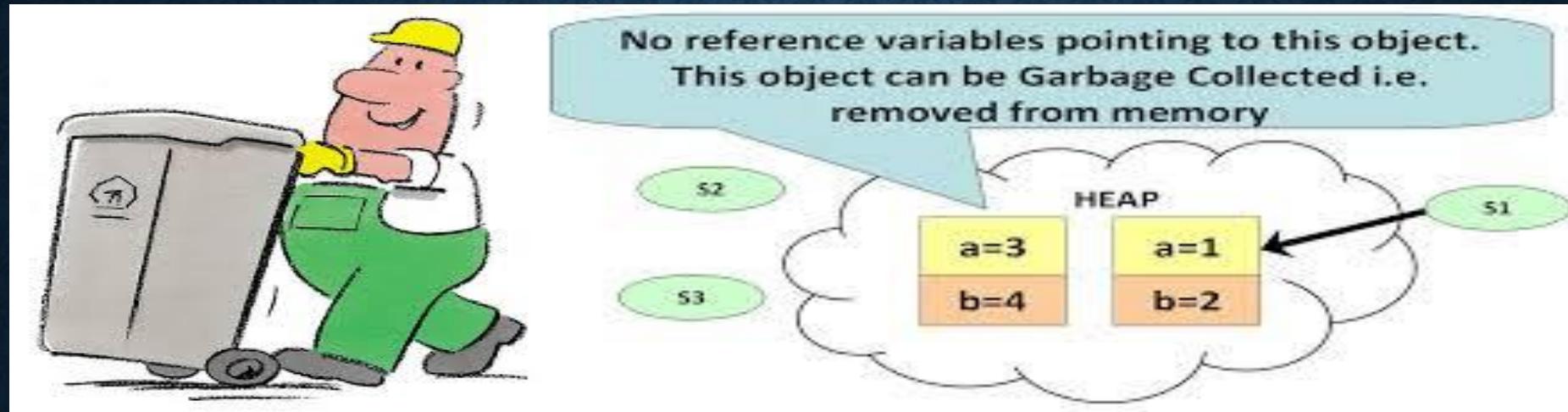
Methods	
Modifier and Type	
protected Object	
clone()	Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

NESTED CLASSES



NESTED CLASS

```
class OuterClass
{
    ...
    static class StaticNestedClass
    {
        ...
    }
    class InnerClass
    {
        ...
        void getResult()
        {
            ...
            class LocalInnerClass
            {
                ...
            }
        }
        ...
    }
}
```

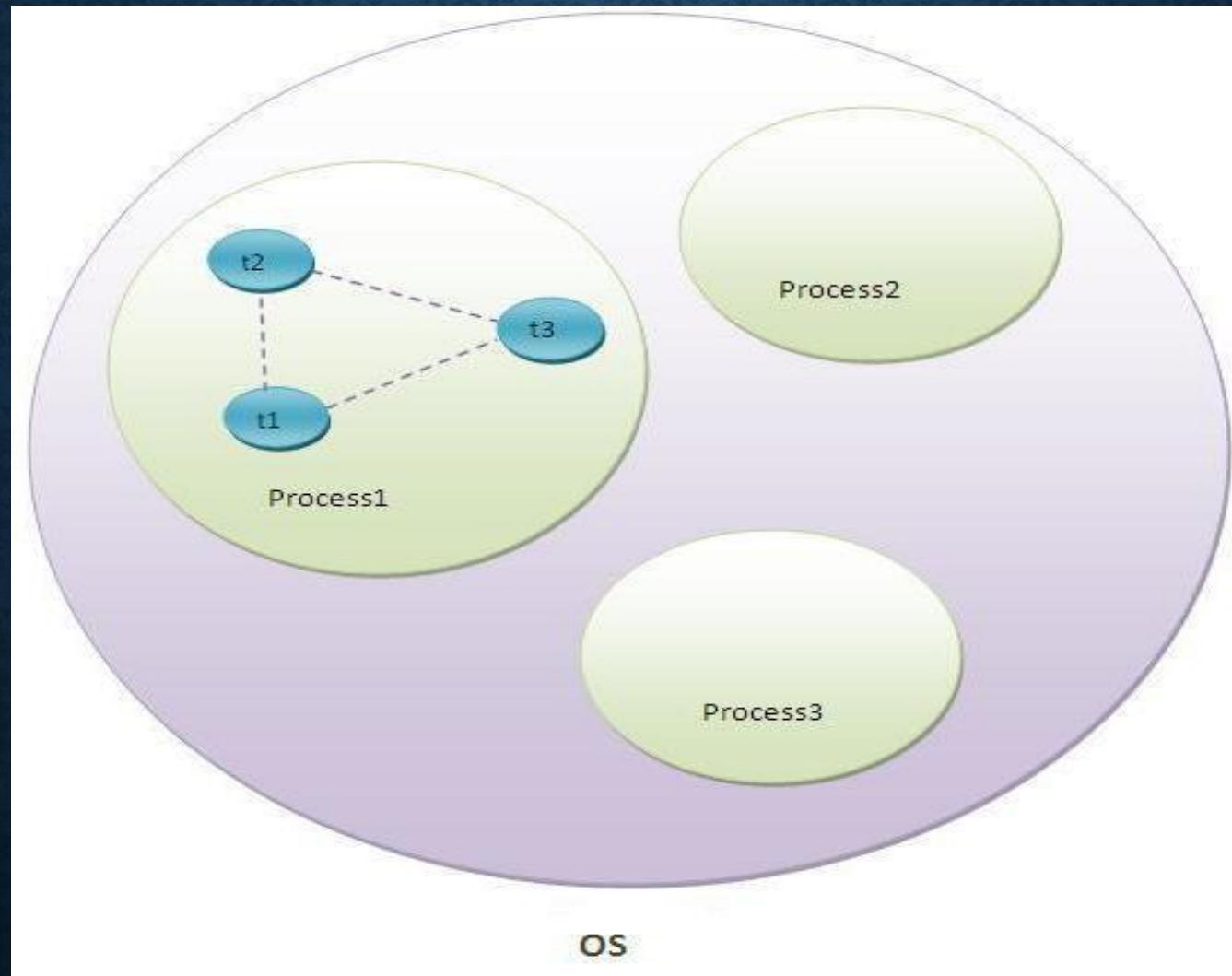


- Garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using `free()` function in C language and `delete()` in C++.
- But, in java it is performed automatically. So, java provides better memory management.

ADVANTAGE OF GARBAGE COLLECTION

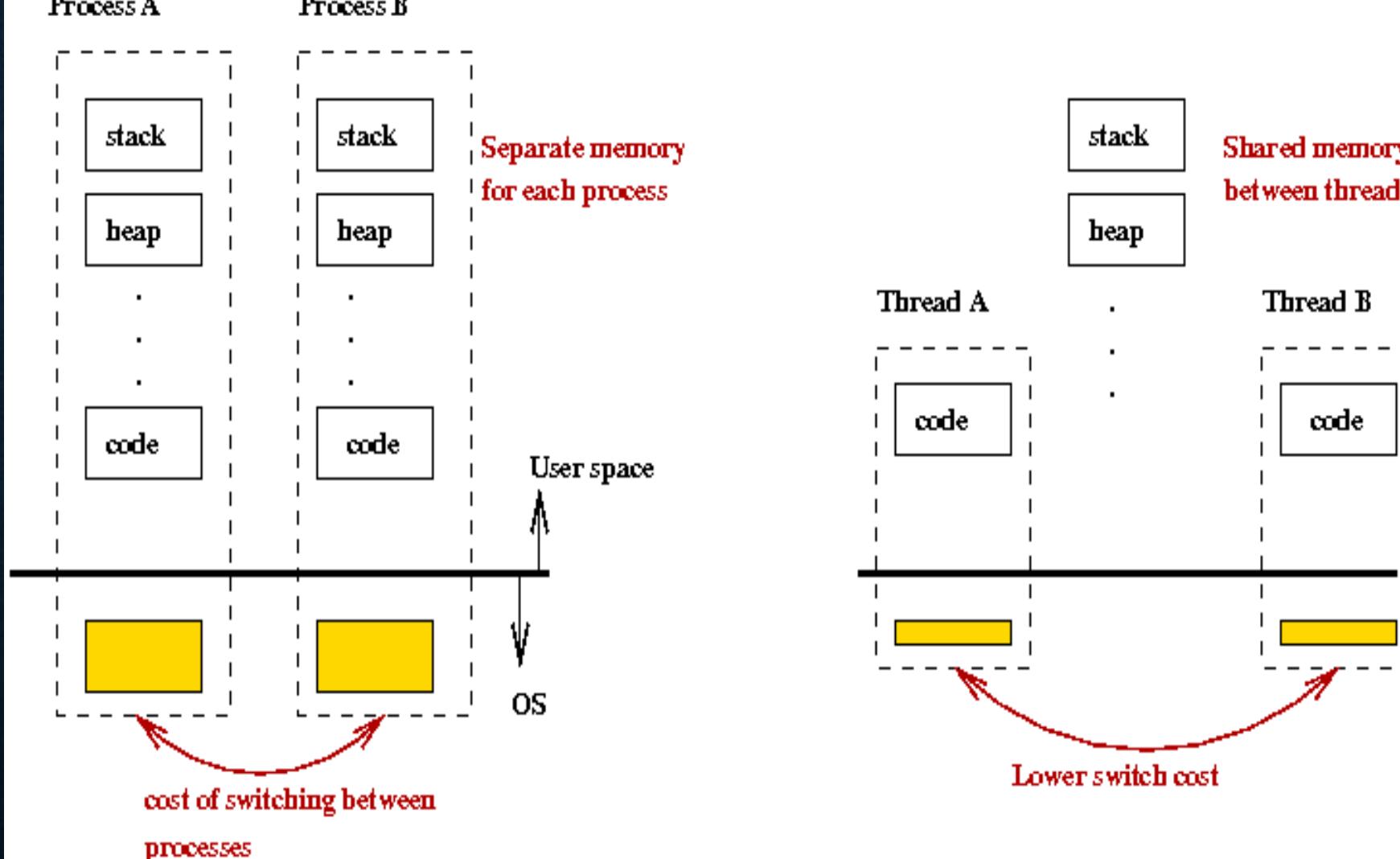
- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.
- A daemon thread called Garbage Collector (GC) automatically performs garbage collection.
- ***This thread calls the finalize() method before object is garbage collected.***
- ***Note: Finalization or garbage collection is not guaranteed.***
-

JAVA PROCESS VS THREADS

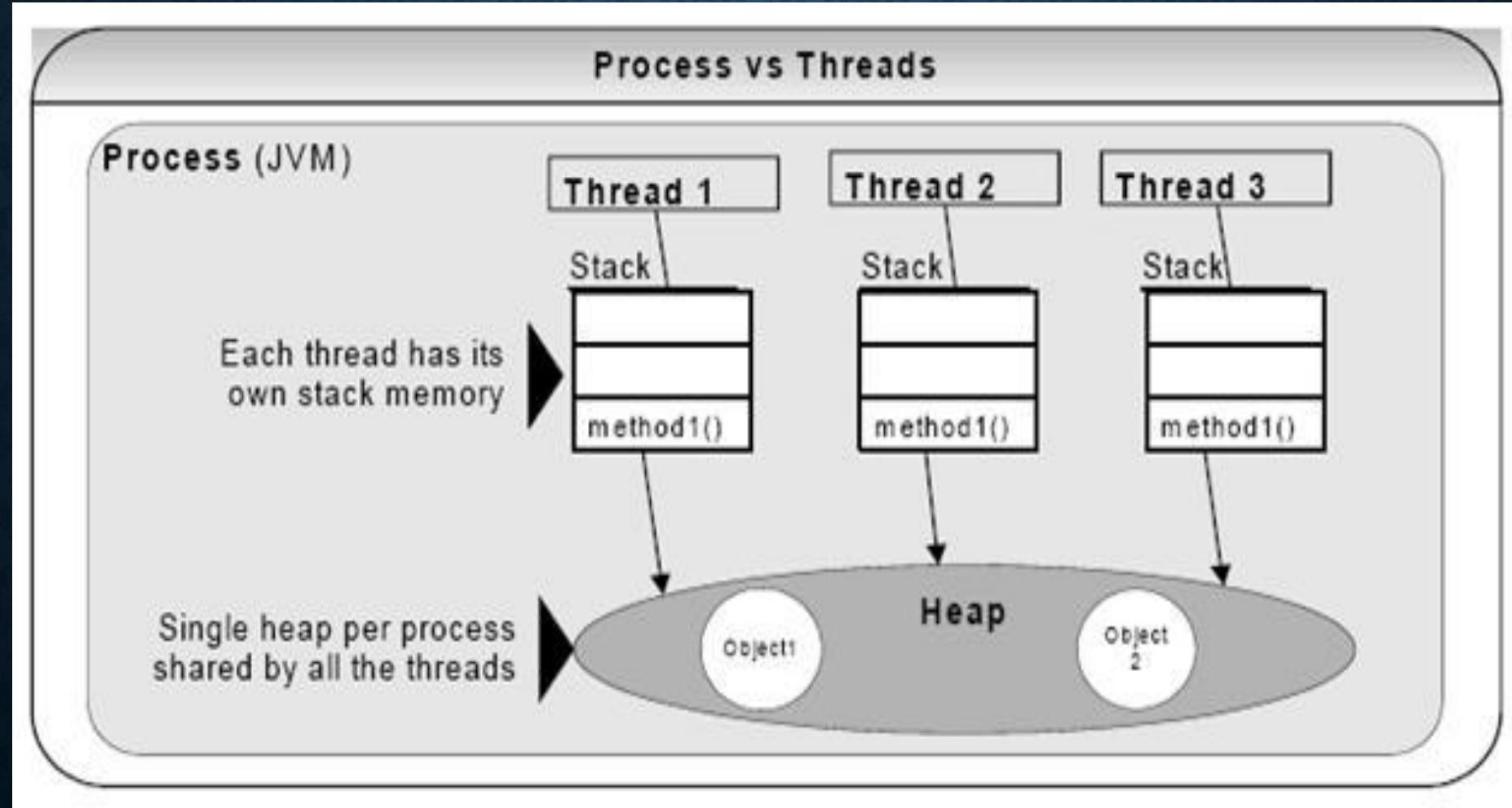


PROCESS VS THREADS

Processes vs. threads



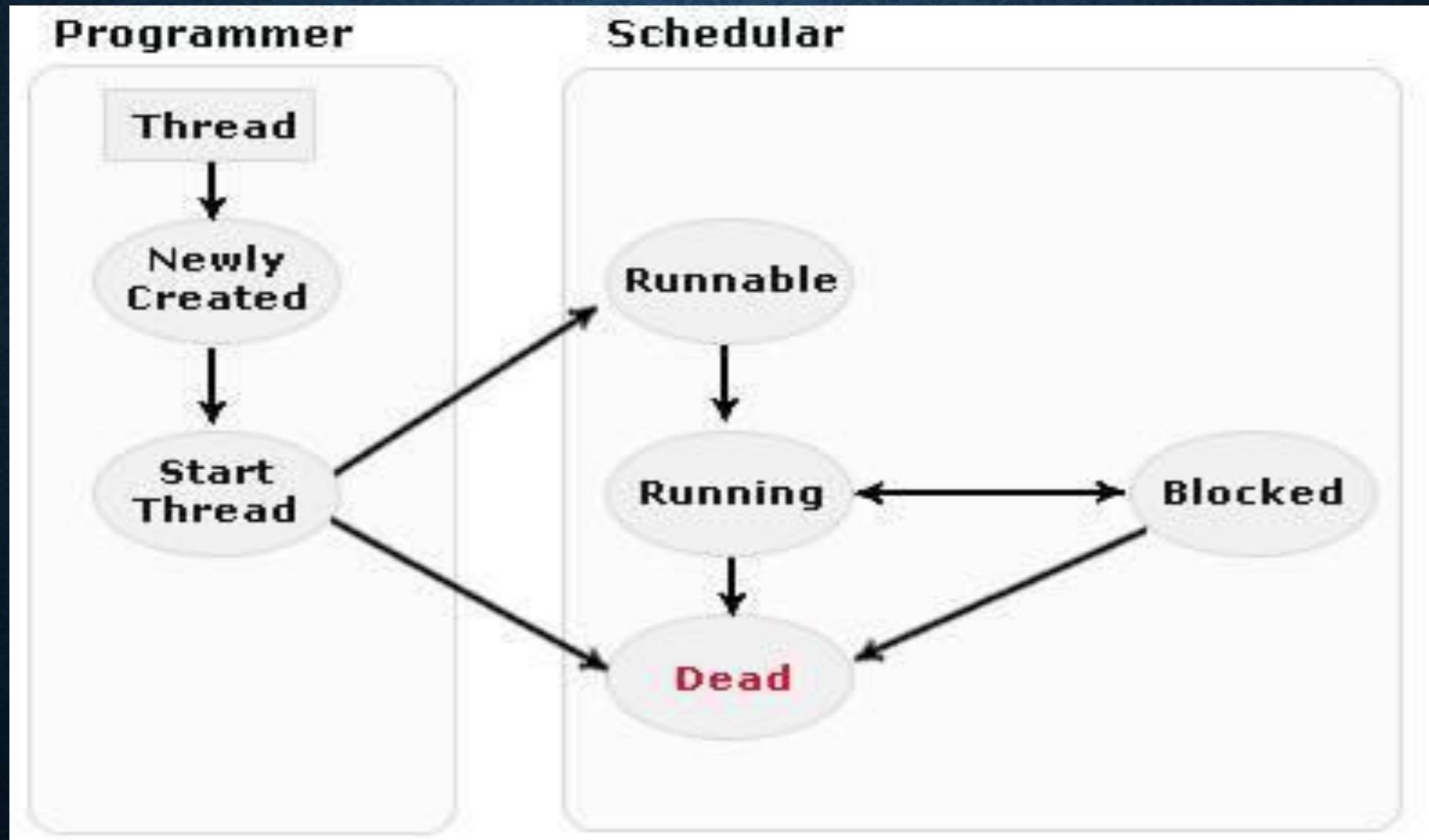
PROCESS VS THREADS



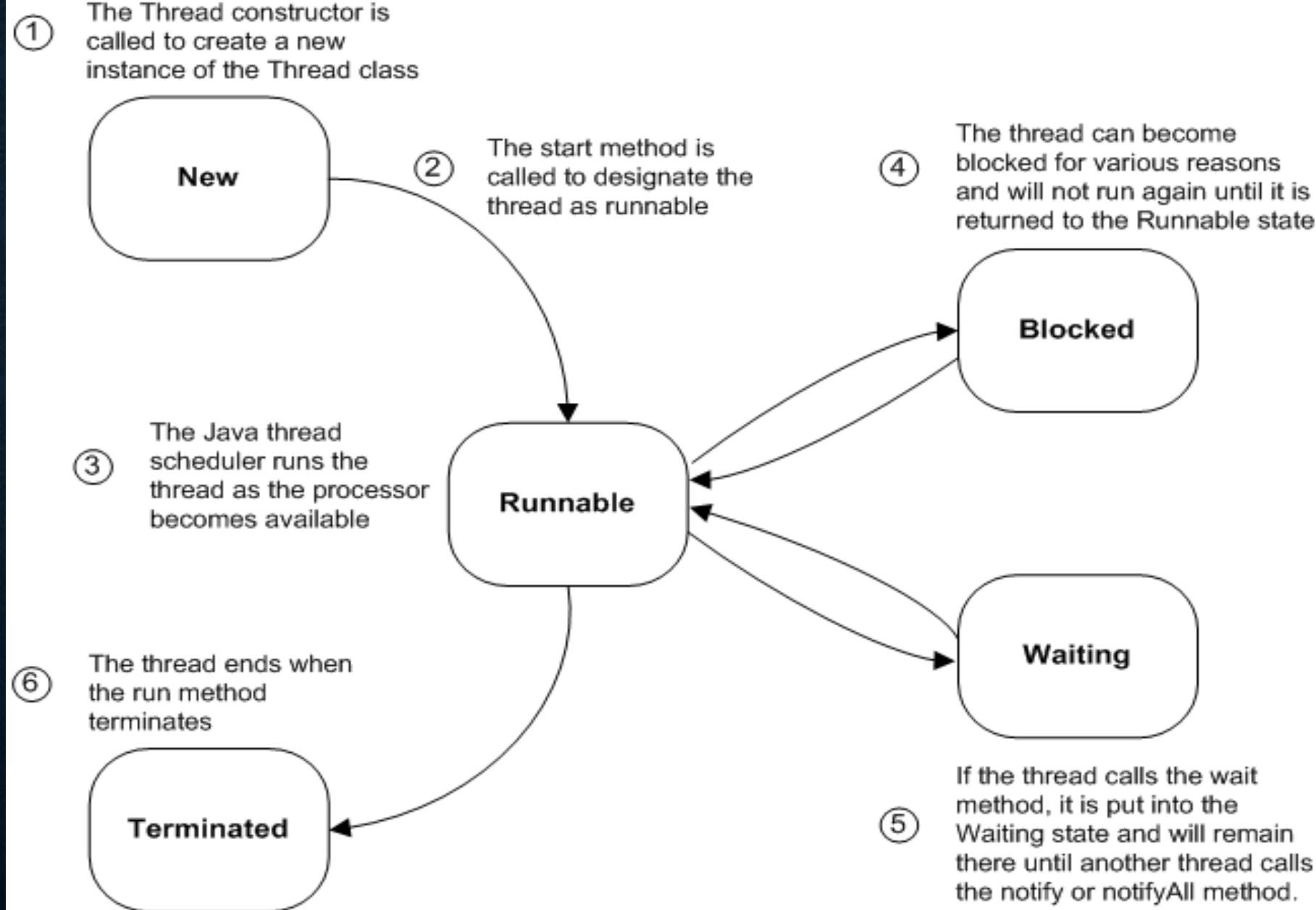
PROCESS VS THREADS

Difference between process and thread?	
Process	Thread
Program in Execution	Separate path of execution One or more threads is called as process
Heavy Weight	Light weight
Requires separate address space	Shares same address space
Inter process communication is expensive.	Inter thread communication is less expensive when compared to process

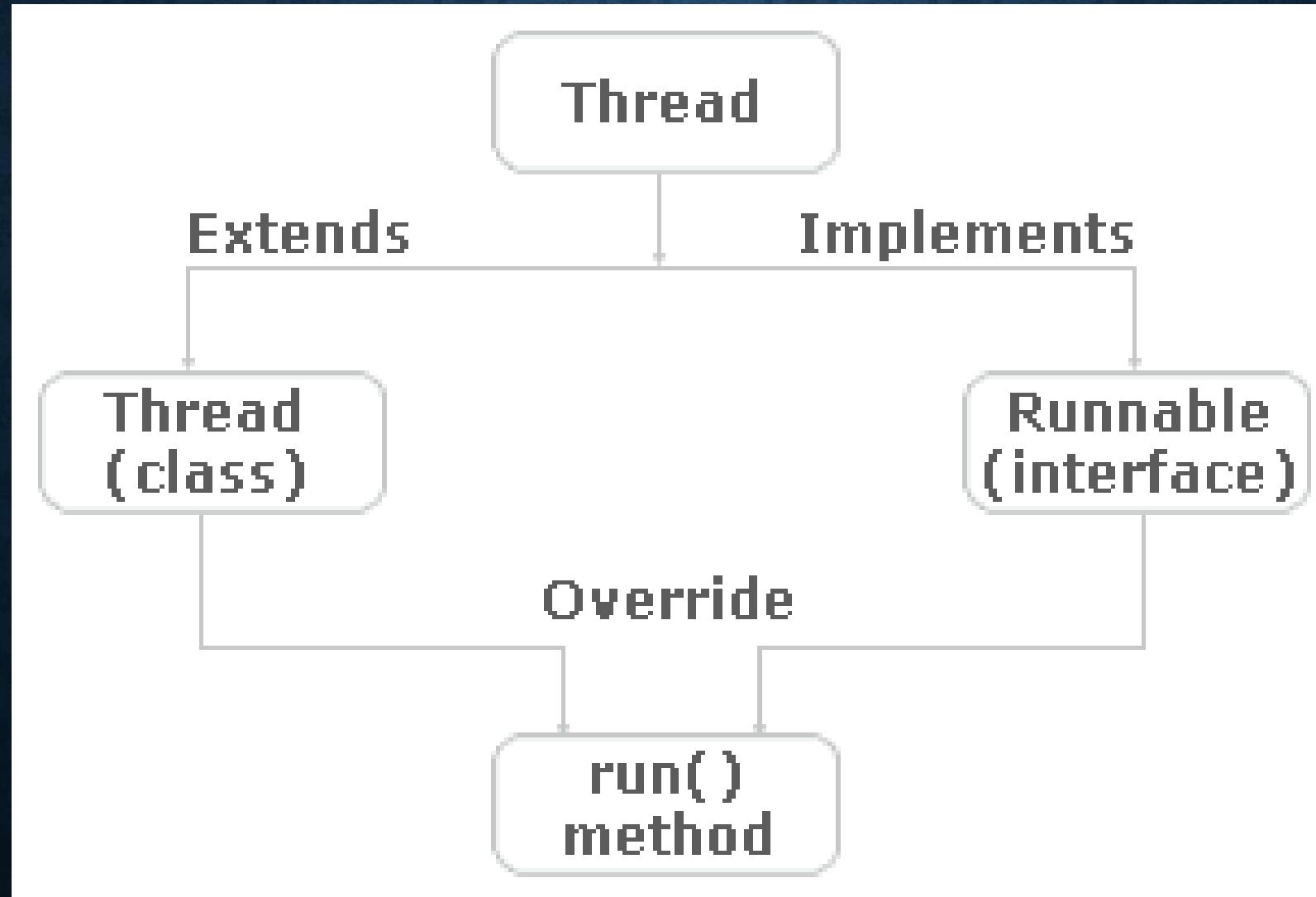
THREAD SCHEDULER



THREAD LIFE CYCLE & STATES



WAYS TO CREATE THREADS



OVERRIDING

RULES FOR METHOD OVERRIDING – PART 1

Clip slide

1. The argument list should be exactly the same as that of the overridden method.
2. The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
3. The access level cannot be more restrictive than the overridden method's access level. For example: if the superclass method is declared public then the overriding method in the sub class cannot be either private or protected.
4. Instance methods can be overridden only if they are inherited by the subclass.
5. A method declared final cannot be overridden.
6. A method declared static cannot be overridden but can be re-declared.

OVERRIDING

RULES FOR METHOD OVERRIDING – PART 2

Clip slide

7. If a method cannot be inherited, then it cannot be overridden.
8. A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
9. A subclass in a different package can only override the non-final methods declared public or protected.
10. An overriding method can throw any unchecked exceptions, regardless of whether the overridden method throws exceptions or not. However the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
11. Constructors cannot be overridden.

JAVA INTERFACE RULES

Member variables

- Can be only public and are by default.
- By default are static and always static
- By default are final and always final

Methods

- Can be only public and are by default.
- Can not be static
- Can not be Final

JDBC STEPS

1. Load the JDBC driver class:

```
Class.forName("driverName");
```

2. Open a database connection:

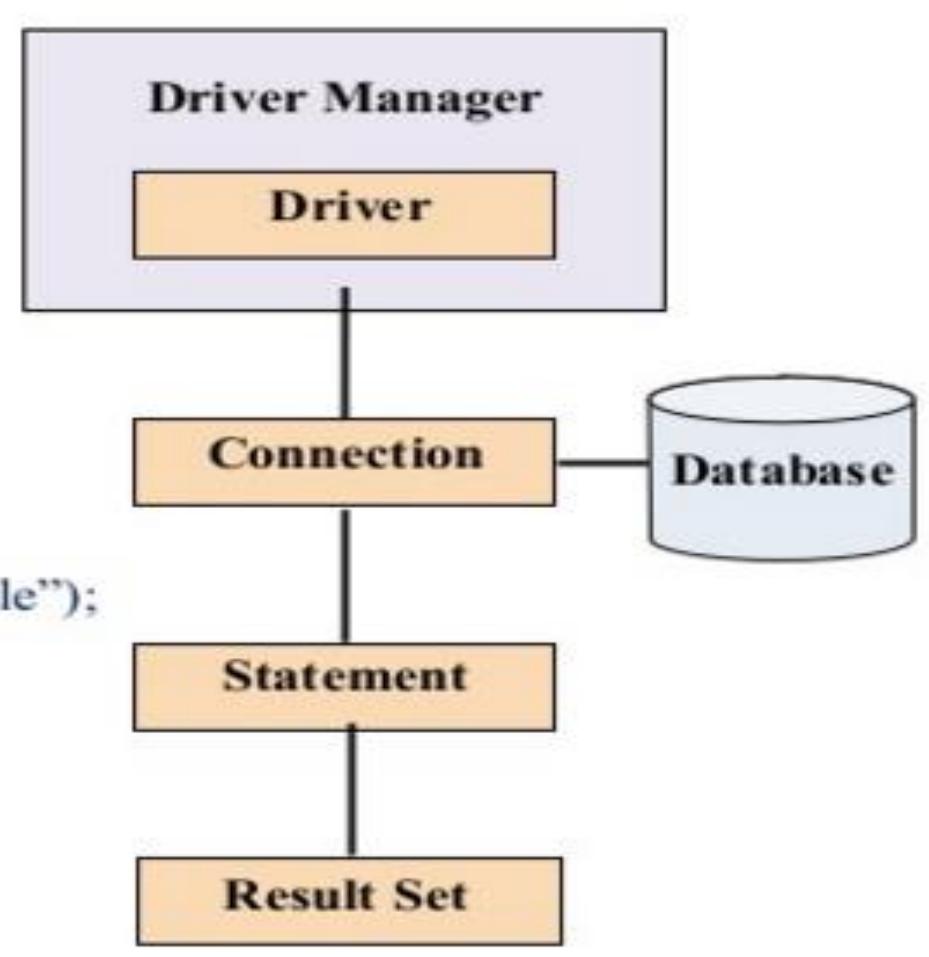
```
DriverManager.getConnection  
("jdbc:xxx:datasource");
```

3. Issue SQL statements:

```
stmt = con.createStatement();  
stmt.executeQuery ("Select * from myTable");
```

4. Process resultset:

```
while (rs.next()) {  
    name = rs.getString("name");  
    amount = rs.getInt("amt"); }
```



ବେଳେ

THE END

