# CS F213 - Object Oriented Programming

Amitesh Singh Rajput

BITS Pilani
Pilani Campus

## Object Class

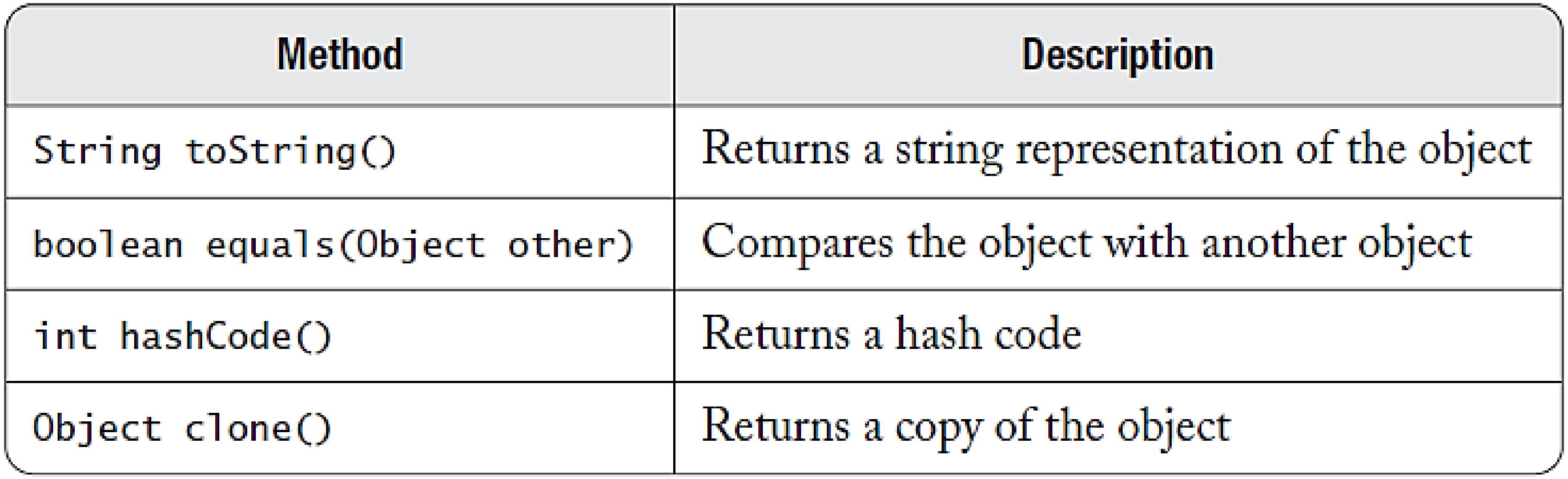
- Every class in Java is directly or indirectly derived from Object class.
  - ➢ If a class extends another class it is indirectly derived

- Methods of the Object class are available to all Java classes.

- It is present in java.lang package.

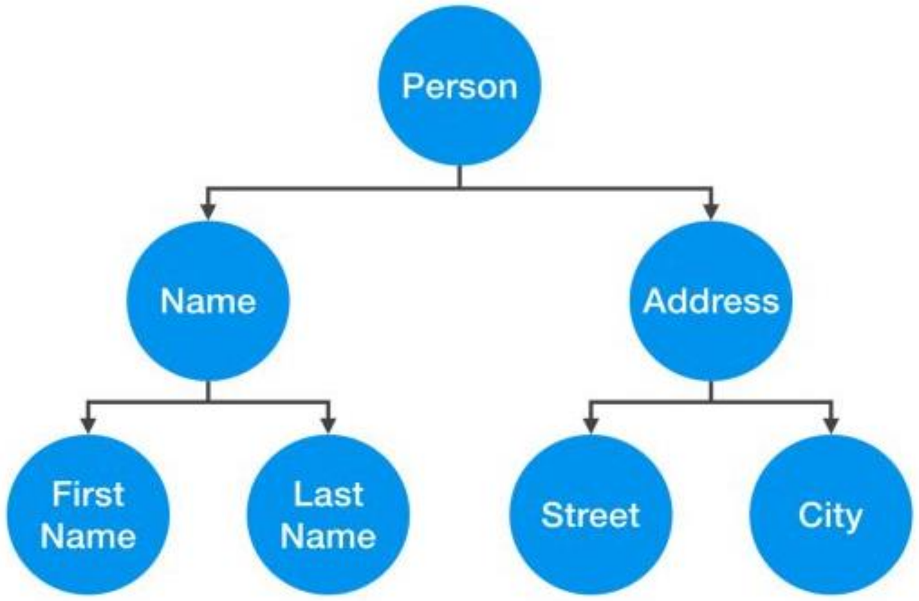
- Object class is the root of inheritance hierarchy.

## Methods in Object Class



| Method | Description |
|---|---|
| `String toString()` | Returns a string representation of the object |
| `boolean equals(Object other)` | Compares the object with another object |
| `int hashCode()` | Returns a hash code |
| `Object clone()` | Returns a copy of the object |

# Objects – An Example

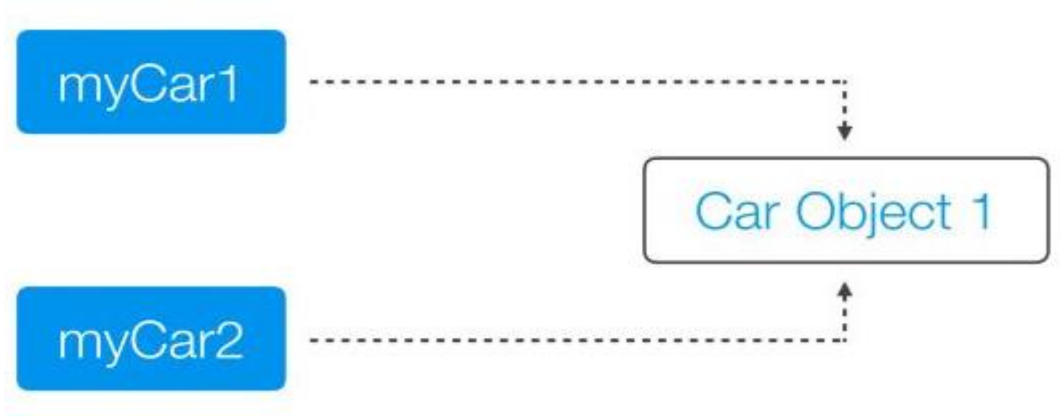


Person is made up of Name and Address objects which in turn are made up of other objects:

- FirstName
- LastName
- Street and
- City

# Copying Objects

- When we use assignment operator it will create a copy of reference variable and not the object.

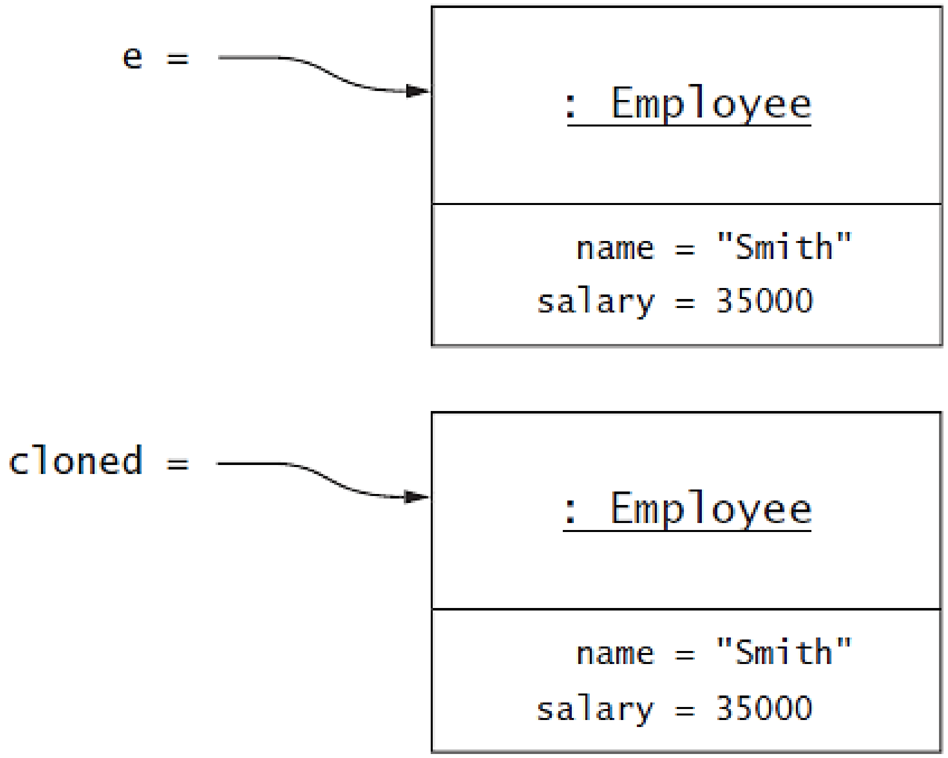


- **Cloning** refers to creation of exact copy of an object.

- It creates a new instance of the class of current object and initializes all its fields with exactly same contents.

# Object cloning

- A deep copy or clone of an object is an object with distinct identity and equal contents.



## Object cloning

- A clone method is expected to fulfill these three conditions:

```
1. x.clone() != x
2. x.clone().equals(x) return true
3. x.clone().getClass() == x.getClass()
```

## Clone Requirements

Any class willing to be cloned must

1. Declare the clone() method to be public
2. Implement Cloneable interface

```
class Account implements Cloneable{
public Object clone(){
    try{
      super.clone()
    }
    catch(CloneNotSupportedException e){ .. }
}
```

When the Object class finds that the object to be cloned isn't an instance of a class that implements Cloneable, it throws a CloneNotSupportedException.

## Example Code

```
public class CloneClassA implements Cloneable{

 @Override
 public CloneClassA clone() throws CloneNotSupportedException{
      Object O1 = super.clone();
      return (CloneClassA) O1;
    }
}
```

## Example Code

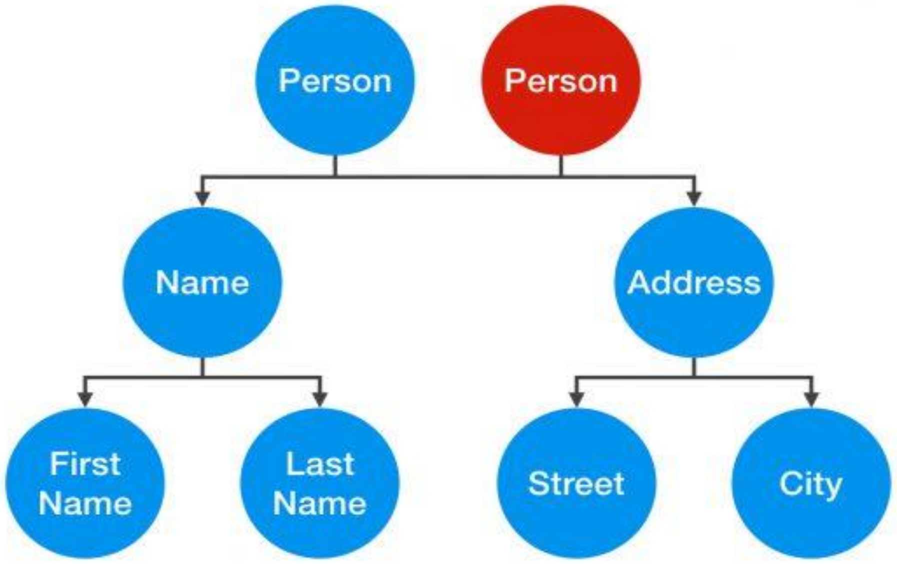
```
public class CloneClassB extends CloneClassA{
 @Override
 public CloneClassB clone() throws CloneNotSupportedException{
   Object O2 = super.clone();
   return (CloneClassB) O2;
 }

 public static void main(String args[]) throws
 CloneNotSupportedException{
   CloneClassB b = new CloneClassB();
   CloneClassB cloned_b = b.clone();
   System.out.println("b: "+ b.hashCode());
   System.out.println("b_cloned: "+ cloned_b.hashCode()); }
}
```

```
b: 483422889
b_cloned: 1209271652
```

## Shallow Copy

- It copies the main object but doesn't copy the inner objects.
- Inner objects are still shared between the original and its copy.

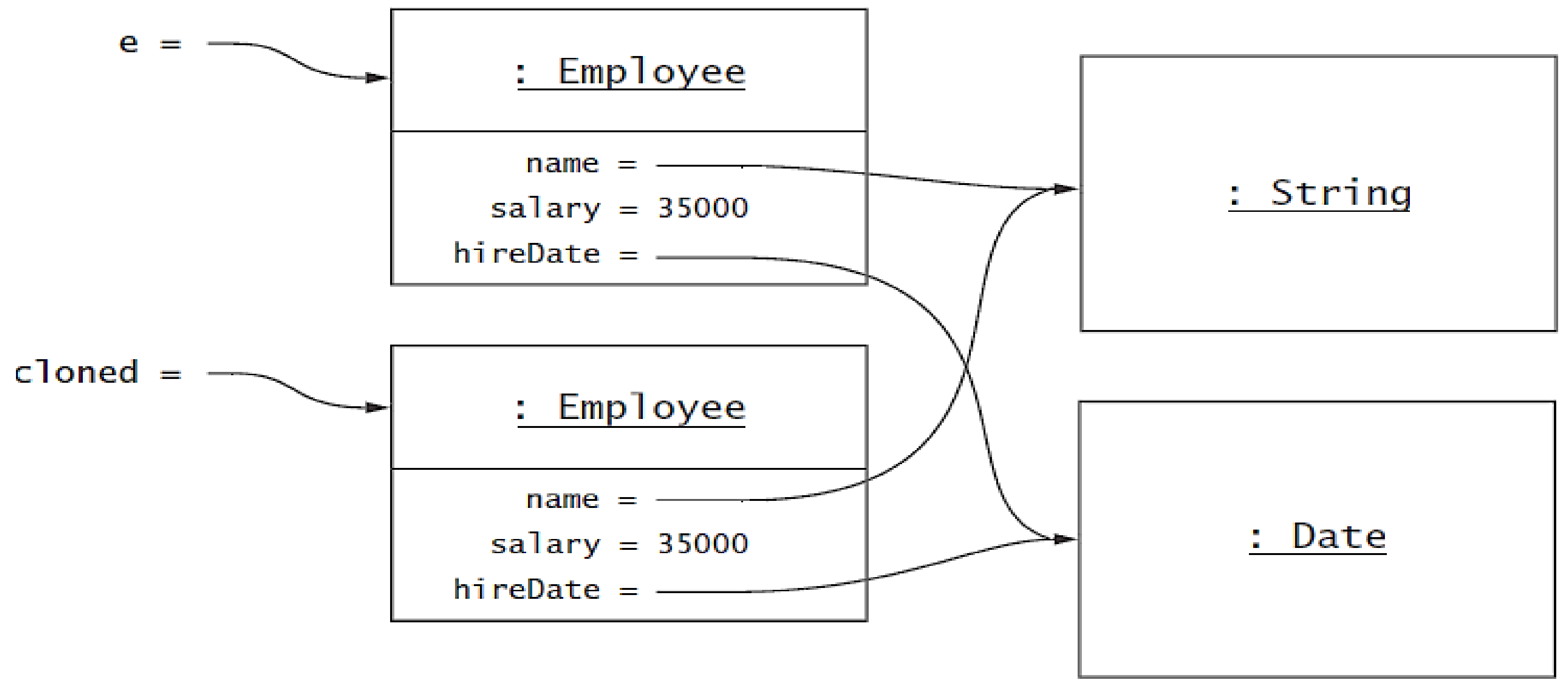


## Shallow Copy - Example

```
public class Employee implements Cloneable{
 public Employee clone(){
  try{
   return (Employee) super.clone();
  }
  catch (CloneNotSupportedException e){
   return null;
  }
 }
 . . .
}
```

- The Object.clone method makes a shallow copy. It makes a new object of the same type as the original and copies the values of all fields.
- If the fields are object references, the original and the clone can share common sub objects.

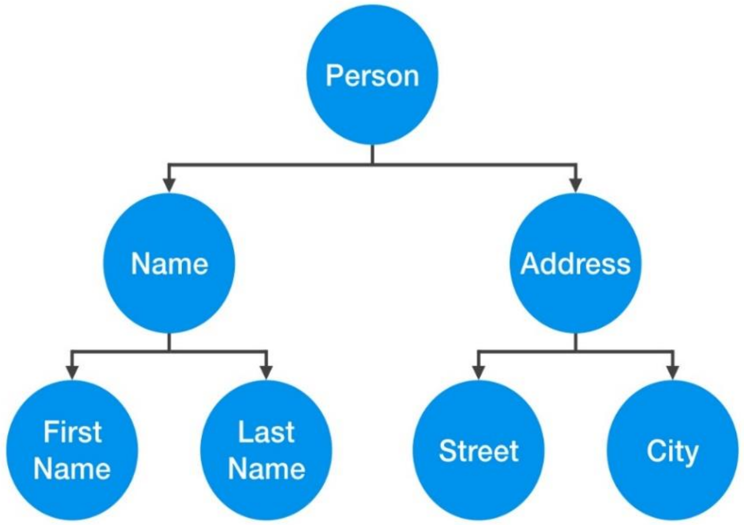
# Shallow Copy - Example

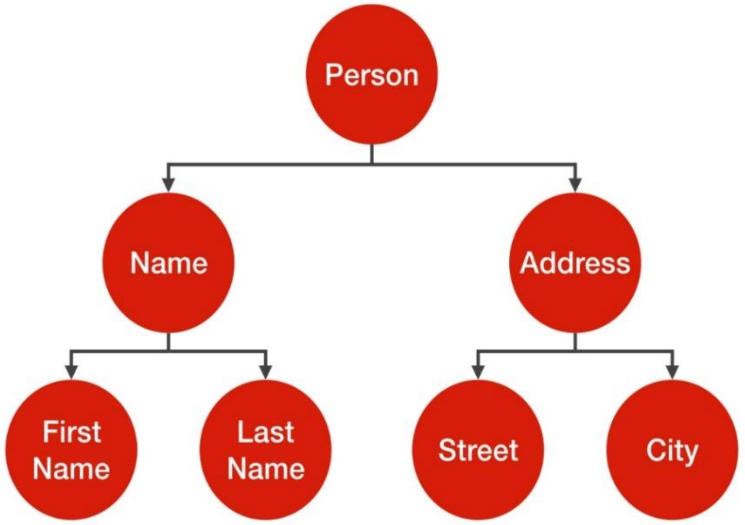


- Sharing of the String object is not a problem as strings are immutable.
- But, sharing a Date is only reasonable if we know that none of the Employee methods mutates it. Otherwise, it too should be cloned.

## Deep Copy

- It is a fully independent copy of an object and it copies the entire object structure



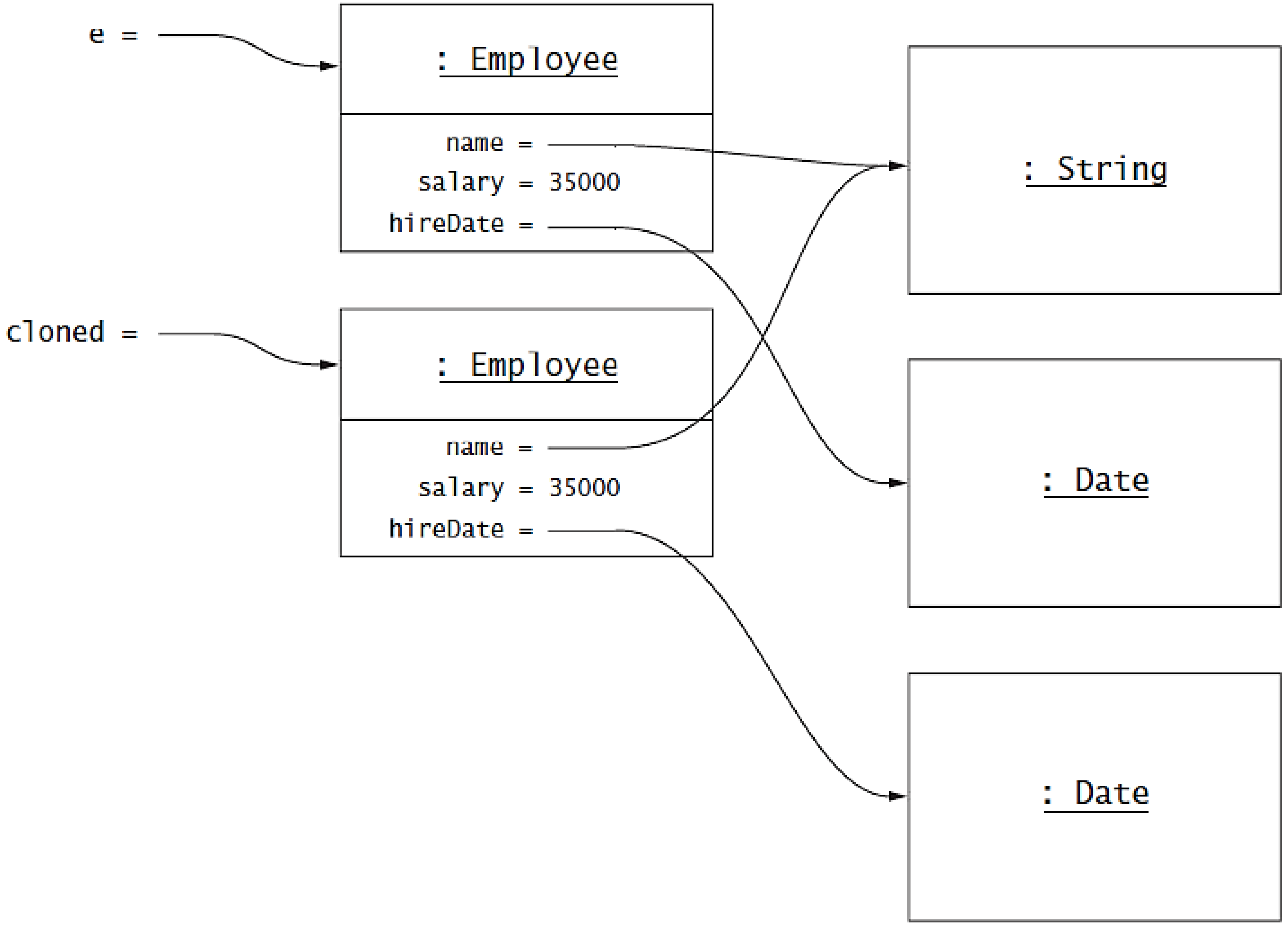


# Deep Copy - Example

```
public class Employee implements Cloneable{

 public Employee clone(){
  try{
    Employee cloned = (Employee) super.clone();
    cloned.hireDate = (Date) hireDate.clone();
    return cloned;
  }
  catch (CloneNotSupportedException e){
   return null;
  }
 }
 . . .
}
```

# Deep Copy - Example



# Java Garbage Collection

- Garbage means unreferenced objects.

- Garbage collection is the process of reclaiming the runtime unused memory automatically.

- Advantage:
  - ➢ Memory efficient because it removes unreferenced objects from heap.
  - ➢ Automatically done by garbage collector.

# When an Object becomes eligible for Garbage Collection?

- If the object is not used by any program, thread, its reference is null.
- If two objects having reference (cyclic reference) of each other and does not have any live reference.
- There are some other cases when an object become eligible for garbage collection:
  - If the reference of that object is explicitly set to null.
  - The object also becomes eligible if it is created inside a block and the reference goes out of the scope once control exit from the block.

# Object Unreferencing

- Nulling the reference
  - `Test t = new Test();`
  - `t = null;`
- Assigning a reference to another
  - `Test t1 = new Test();`
  - `Test t2 = new Test();`
  - `t1 = t2;`
  - Now the first object reference by **`t1`** is available for garbage collection
- By anonymous object
  - `new Test();`

- **finalize()**
  - This method is called before garbage collection when an object has no more references.
  - It could be overridden to dispose system resources, perform clean up and minimize memory leaks.
  - finalize() method is called just once on an object.
  - protected void finalize().

- **gc()**
  - It is used to invoke the garbage collector to perform clean up.
  - It is found in System and Runtime classes.
  - public static void gc().

## Java Runtime class

- It is used to interact with the Java runtime environment.
- It provides methods to execute a process, invoke GC, get total and free memory, etc.
- Only one instance of the java.lang.Runtime class is available for one Java application.

# Java Runtime class - Example

**public long freeMemory():** Returns the amount of free memory in the JVM.

```
public class Abc{
 public static void main(String[] args){
  System.out.println(""+Runtime.getRuntime().freeMemory());
 }
}
```

```
266421656
```

## Java Runtime class - Example

**public long totalMemory():** Returns the amount of total memory in the JVM.

```
public class Abc{
 public static void main(String[] args){
  System.out.println(""+Runtime.getRuntime().totalMemory());
 }
}
```

```
268435456
```

# Java Runtime class - Example

Checking the amount of used memory and converting it to MB.

```
public class Abc{
 public static void main(String[] args){
  System.out.println(""+Runtime.getRuntime().freeMemory());
  System.out.println(""+Runtime.getRuntime().totalMemory());
  long b = Runtime.getRuntime().totalMemory()-
                       Runtime.getRuntime().freeMemory();
  System.out.println(""+b/(1024*1024));
 }
}
```

```
1
```

innovate achieve lead

BITS Pilani
Pilani Campus

# Thank You!