Methods inherited from Object

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Return | Params. | Desc. |
| Public toString | String | Void |  |
| Public equals | boolean | Object obj |  |
| Public hashCode | Int | Void |  |

**Equals Contract**

1. **Reflexivity: Object must be equal to itself.**
2. **Symmetry: Any two objects must agree on how they are equal. For any non-null reference values x and y, x.equals(y) returns true if and only if y.equals(x) returns true.**

**Example:**

**public class** CaseInsensitiveString{  
 **private final** String **s**;  
  
 **public** CaseInsensitiveString(String s){  
 **if**(s == **null**)  
 **throw new** NullPointerException();  
 **this**.**s** = s;  
 }  
 **public boolean** equals(Object o){  
 **if**(o **instanceof** CaseInsensitiveString)  
 **return s**.equalsIgnoreCase(((CaseInsensitiveString) o).**s**);  
  
 **if**(o **instanceof** String)  
 **return s**.equalsIgnoreCase((String) o);  
 **return false**;  
 }  
  
  
 **public static void** main (String[] args){  
 CaseInsensitiveString cis = **new** CaseInsensitiveString(**"OCJP"**);  
 String s = **"OCJP"**;  
 **assert** (cis.equals(s));  
 **assert (!s.equals(cis)); // SYMMETRY BROKEN**  
 }  
 }

CaseInsensitiveString knows it can handle ordinary Strings in its equals method, but regular String does not know anything about CaseInsensitiveString.

1. Transitivity: For any non-null reference values x,y,z, if x.equals(y) returns true and y.equals(z) returns true , then x.equals(z) must return true.

**Example:**

**public class** Point {  
  
 **private final int x**;  
 **private final int y**;  
  
 **public** Point(**int** x, **int** y){  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
  
 @Override  
 **public boolean** equals(Object o){  
 **if**(!(o **instanceof** Point))  
 **return false**;  
 Point p = (Point)o;  
 **return** p.**x** == **x** && p.**y** == **y**;  
 }  
}

**public class** ColorPoint **extends** Point {  
  
 **private final** COLOR **color**;  
 **public** ColorPoint(**int** x, **int** y, COLOR color){  
 **super**(x,y);  
 **this**.**color** = color;  
 }  
  
 @Override  
 **public boolean** equals(Object o){  
 **if**(!(o **instanceof** Point))  
 **return false**;  
 *// if o is a normal Point, do a color-blind comparision* **if**(!(o **instanceof** ColorPoint))  
 **return** o.equals(**this**);  
  
 *// if o is a colorPoint; do a full comparision* **return super**.equals(o) && ((ColorPoint) o).**color** == **color**;  
 }  
  
 **public static void** main(String[] args){  
 ColorPoint p1 = **new** ColorPoint(1,2,COLOR.***BLUE***);  
 Point p2 = **new** Point(1,2);  
 ColorPoint p3 = **new** ColorPoint(1,2,COLOR.***RED***);  
  
 **assert** (!p1.equals(p3)); ***// BREAKS Transitivity , even though Point and Color Point are equal because it does a Point comparision, Color Point comparision returns false.*****assert** (p1.equals(p2));  
 **assert** (p3.equals(p2));  
 }  
  
  
}

**Hashcode Contract:**

1. Whenever it is invoked on the same object more than once during an execution of an application, the hashCode method must consistently return the same integer, provided no information used in equals comparison on the object is modified. The integer need not remain consistent from one execution of an application to another execution of the same application.
2. If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
3. It is not required that if two objects are unequal according to the equals(Object) method, then the calling hashCode method on each of the two objects must produce a distinct integer result. However, programmer should be aware that producing distinct integer result for unequal objects may improve performance of hash tables.

**Types of nested classes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Member Inner Class** | **Local Inner Class** | **Anonymous Inner class** | **Static nested class** |
| **Access modifiers allowed** | Public, protected, private, or default access | None, Already local to method | None, Already local to statement | Public, protected, private or default access |
| **Can extend any class and any number of interfaces** | Yes | Yes | No- must have exactly one super class or one interface | Yes |
| **Can be abstract** | Yes | Yes | N/A because no class definition | Yes |
| **Can be final** | Yes | Yes | N/A because no class definition | Yes |
| **Can access instance members of enclosing class** | Yes | Yes | Yes | No (not directly; requires an instance of the enclosing class) |
| **Can access local variables of enclosing class** | No | Yes- if final or effectively final | Yes – if final or effectively final | No |
| **can declare static methods** | No | No | No | Yes |