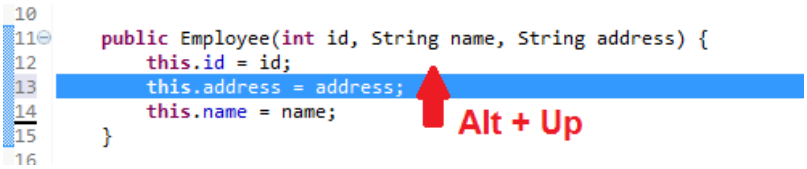
27 Eclipse Shortcut Keys for Code Editing

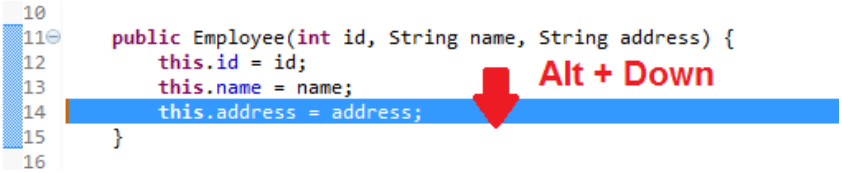
When using an IDE, you cannot be more productive without using its shortcut keys frequently as your habit. In this article, we summarize a list of shortcut keys which are useful for editing Java code in Eclipse IDE.

NOTE: Standard shortcuts are not covered, such as Ctrl + A (select all), Ctrl + Z (undo), etc.

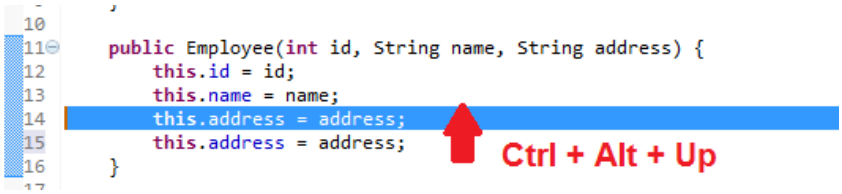
1. **Ctrl + D**: Deletes current line.
2. **Ctrl + Delete:** Deletes next word after the cursor.
3. **Ctrl + Shift + Delete:** Deletes from the cursor until end of line.
4. **Ctrl + Backspace:** Deletes previous word before the cursor.
5. **Shift + Ctrl + y:** Changes a selection to lowercase.
6. **Shift + Ctrl + x:** Changes a selection to uppercase.
7. **Alt + Up Arrow**: Moves up current line (or a selected code block) by one line:



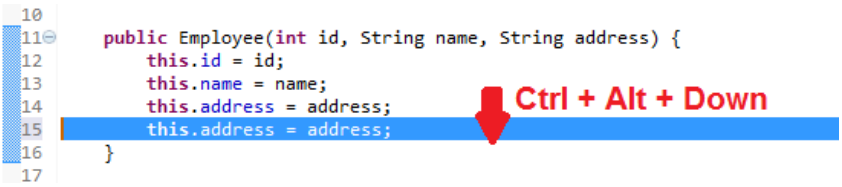
8. **Alt + Down Arrow**: Moves down current line (or a selected code block) by one line:



9. **Ctrl + Alt + Up Arrow:** Copies and moves up current line (or a selected code block) by one line:



10. **Ctrl + Alt + Down Arrow:** Copies and moves down current line (or a selected code block) by one line:



11. **Shift + Enter:** Inserts a blank line after current line, regardless where the cursor is at the current line (very different from press **Enter** key alone):

```
    public Employee(int id, String name, String address) {
        this.id = id;
        this.name = name;

        this.address = address;
    }
```

Shift + Enter

12. **Ctrl + Shift + Enter:** works similar to the **Shift + Enter**, but inserts a blank line just before the current line.
13. **Ctrl + Shift + O:** Organizes import statements by removing unused imports and sorts the used ones alphabetically. This shortcut also adds missing imports.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.servlet.ModelAndView;

import java.util.List;
import java.util.ArrayList;

import com.ava.hr.dao.EmployeeDAO;
import com.ava.hr.dao.UserDAO;
import com.ava.hr.model.Employee;
import com.ava.hr.model.User;
```

Ctrl + Shift + O to remove these unused imports

14. **Ctrl + Shift + M:** Adds a single import statement for the current error due to missing import. You need to place the cursor inside the error and press this shortcut:

```
    @RequestMapping(value="/")
    public ModelAndView viewIndex() {
        ModelAndView model = new ModelAndView("index");
        model.addObject("user", new User());
        return model;
    }
```

Ctrl + Shift + M to add import for the User class

15. **Ctrl + Shift + F:** Formats a selected block of code or a whole source file. This shortcut is very useful when you want to format messy code to Java-standard code. Note that, if nothing is selected in the editor, Eclipse applies formatting for the whole file:

```
public class EmployeeDAO {
private SessionFactory sessionFactory;

        public EmployeeDAO(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional
public void save(Employee employee)
    {
    Session session = sessionFactory.getCurrentSession();

        session.save(employee);
    }

}
```

Ctrl + Shift + F to format this messy code

16. **Ctrl + I:** Corrects indentation for current line or a selected code block. This is useful as it helps you avoid manually using **Tab** key to correct the indentation:

```
26    @RequestMapping(value = "/")
27    public ModelAndView viewIndex() {
28            ModelAndView model = new ModelAndView("index");
29        model.addObject("user", new User());
30        return model;
31    }
```

Press Ctrl + I to correct indentation of this line

17. **Ctrl +A, Ctrl+ I:** Corrects indentation for hold page.
18. **Ctrl + /**or **Ctrl + 7**: Toggle single line comment. This shortcut adds single-line comment to current line or a block of code. Press again to remove comment. For example:

```
3  import org.hibernate.Session;
8
9  public class EmployeeDAO {
10     private SessionFactory sessionFactory;
11
12     public EmployeeDAO(SessionFactory sessionFactory) {
13         this.sessionFactory = sessionFactory;
14     }
15
16 //  @Transactional
17 //  public void save(Employee employee) {
18 //      Session session = sessionFactory.getCurrentSession();
19 //
20 //      session.save(employee);
21 //  }
22
23 }
24
```

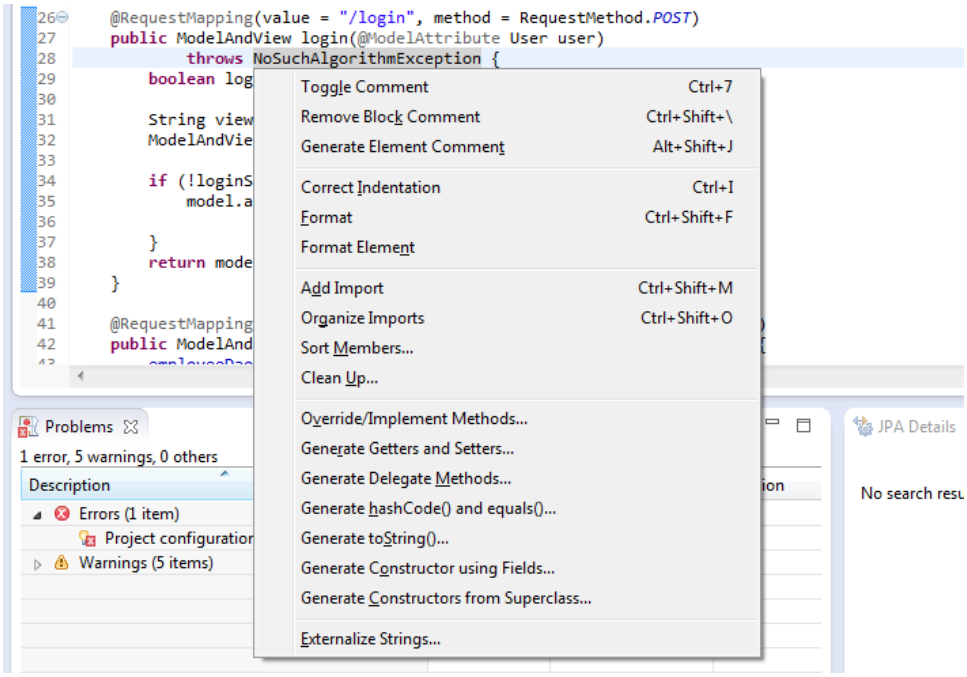
Ctrl + / or Ctrl + 7

19. **Ctrl + \:** Single line comment.
20. **Ctrl + Shift + /:** Adds block comment to a selection.

```
9  public class EmployeeDAO {
10     private SessionFactory sessionFactory;
11
12     public EmployeeDAO(SessionFactory sessionFactory) {
13         this.sessionFactory = sessionFactory;
14     }
15
16 /*  @Transactional
17     public void save(Employee employee) {
18         Session session = sessionFactory.getCurrentSession();
19
20         session.save(employee);
21     }*/
22
23 }
24
```

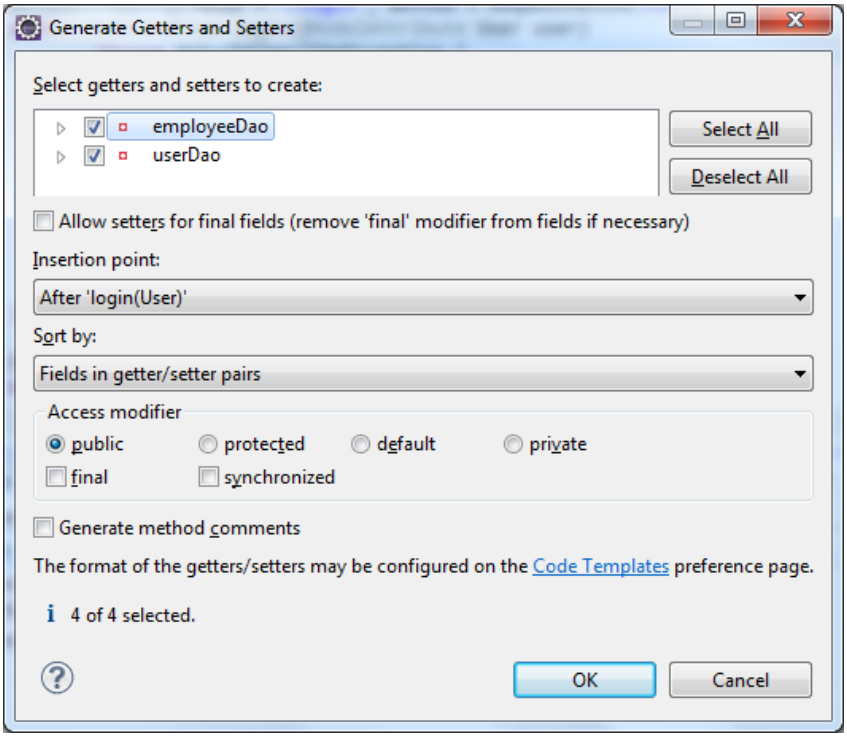
Ctrl + Shift + / to add a block comment

21. **Ctrl + Shift + \:** Removes block comment.
22. **Alt + Shift + S:** Shows context menu that lists possible actions for editing code:

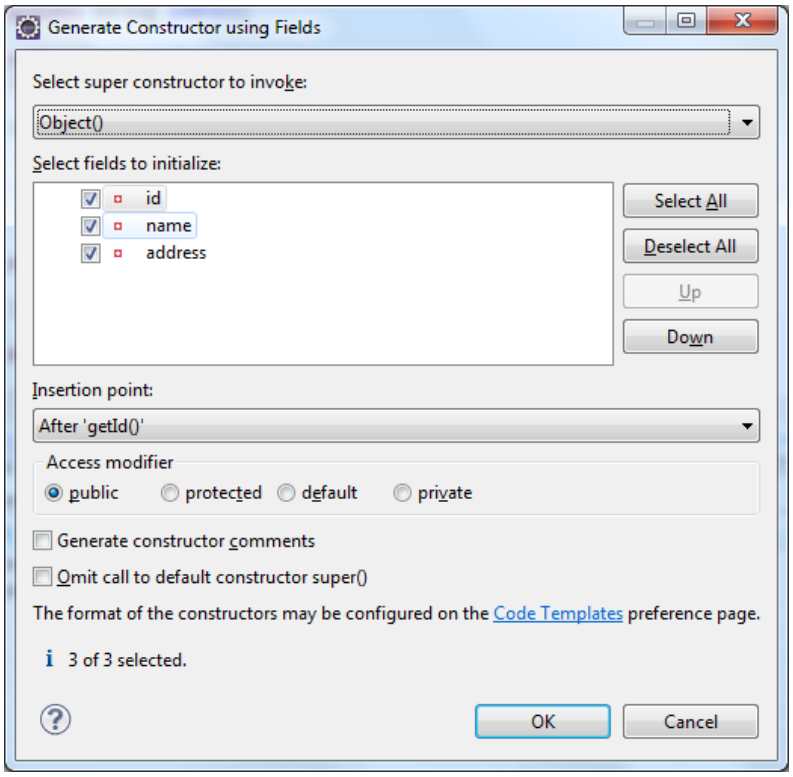


From this context menu, you can press another letter (according to the underscore letters in the names) to access the desired functions.

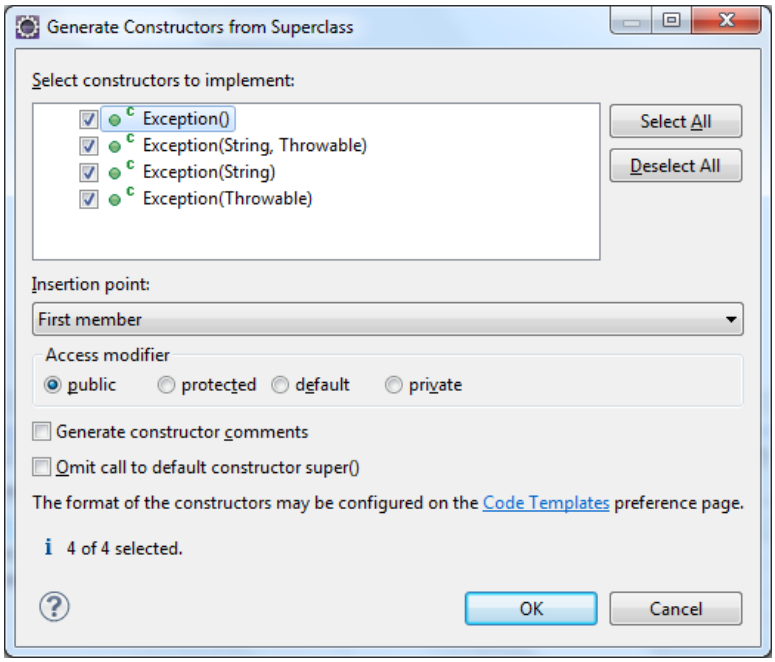
23. **Alt + Shift + S, R:** Generates getters and setters for fields of a class. This is a very handy shortcut that helps us generate getter and setter methods quickly. The following dialog appears:



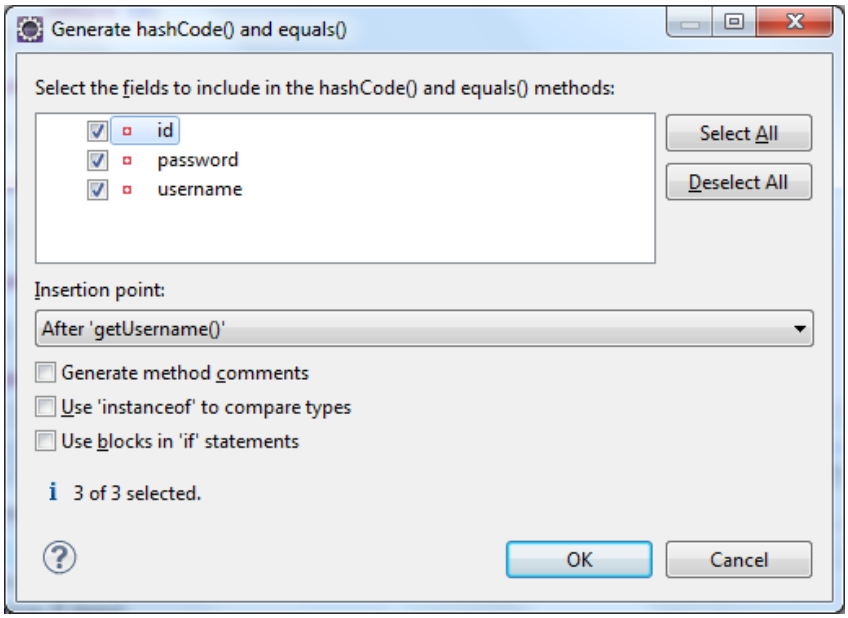
24. **Alt + Shift + S, O:** Generates constructor using fields. This shortcut is very useful when you want to generate code for a constructor that takes class' fields as its parameters. The following dialog appears:



25. **Alt + Shift + S, C:** Generates Constructors from Superclass. A common example for using this shortcut is when creating a custom exception class. In this case, we need to write some constructors similar to the `Exception` superclass. This shortcut brings the *Generate Constructors from Superclass* dialog which allows us to choose the constructors to be implemented in the subclass:



26. **Alt + Shift + S, H:** Generates hashCode() and equals() methods, typically for a JavaBean/POJO class. The class must have non-static fields. This shortcut brings the *Generate hashCode() and equals()* dialog as below:



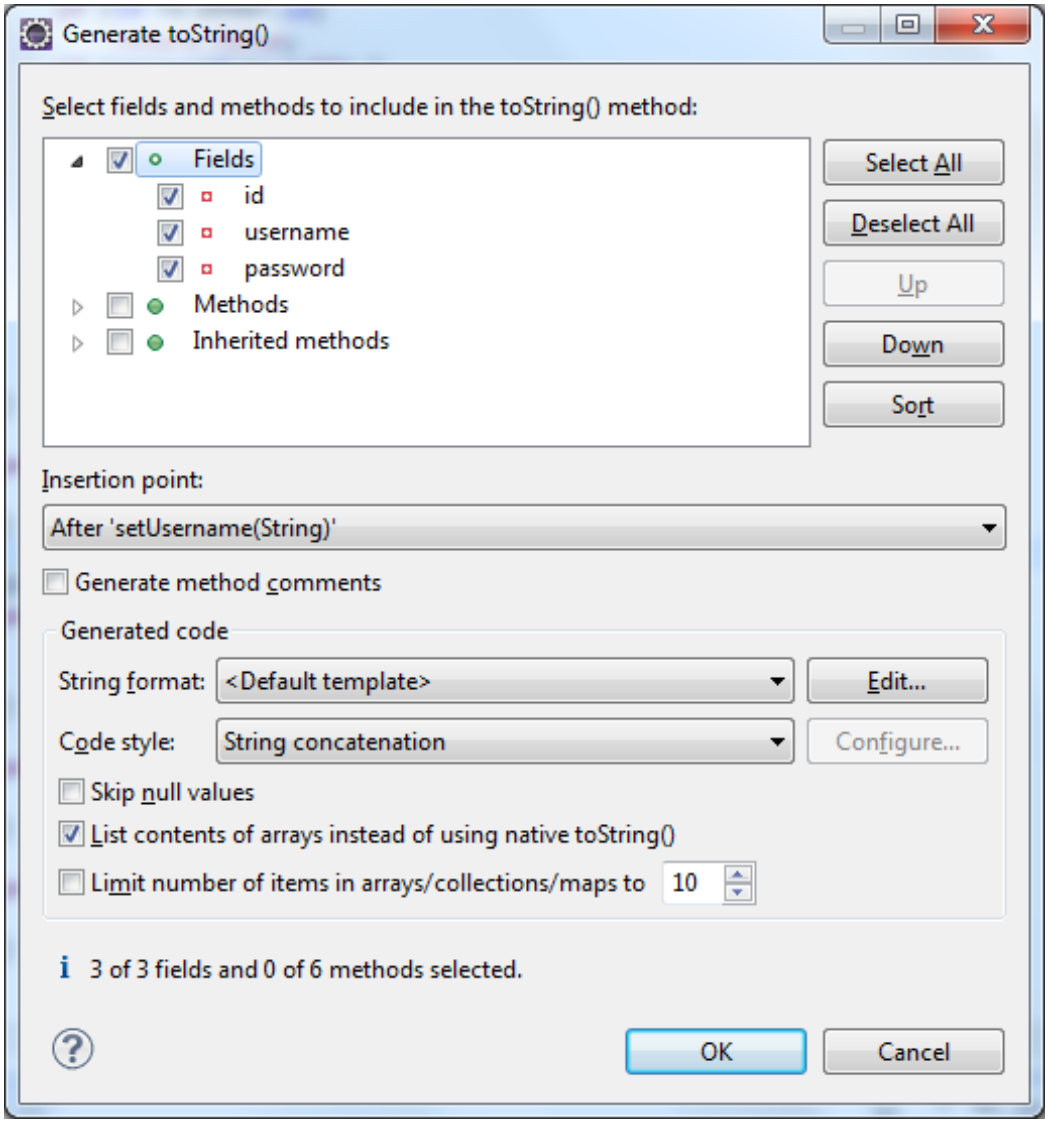
Select the fields to be used in **`hashCode()`** and **`equals()`** method, and then click OK. We got the following result (example):

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + id;
    result = prime * result
            + ((password == null) ? 0 : password.hashCode());
    result = prime * result
            + ((username == null) ? 0 : username.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    User other = (User) obj;
    if (id != other.id)
        return false;
    if (password == null) {
        if (other.password != null)
            return false;
    } else if (!password.equals(other.password))
        return false;
    if (username == null) {
        if (other.username != null)
            return false;
    } else if (!username.equals(other.username))
```

```
            return false;
        return true;
    }
```

27. **Alt + Shift + S, S:** Generates `toString()` method. This shortcut comes in handy if we want to override `toString()` method that returns information of relevant fields of the class. This brings the *Generate toString()* dialogas below:



And here's sample of a generated `toString()` method:

```
@Override
public String toString() {
    return "User [id=" + id + ", username=" + username + ", password="
            + password + "]";
}
```