JAVA BASICS

Functional Programming in Java, Generics and Annotations & more…

Java workshop training, August,2017.

Dr. Kishore Biswas (Forrest/柯修)

PhD. Artificial Intelligence—Natural Language Processing.

CanWay IT Training ®.

Contents:

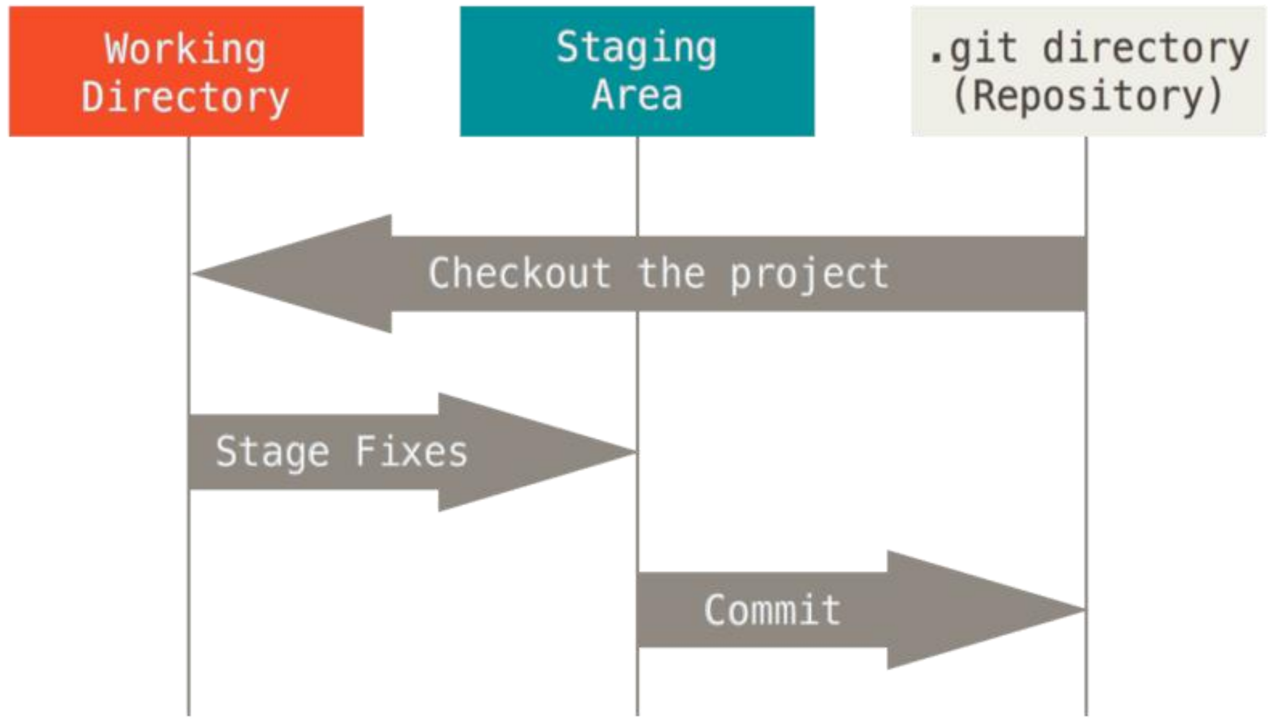
- 1) Weekly test!
- 2) GIT & GIT hub (Continued…)
- 3) Principle for Usable Design & Usability Metrics.
- 4) Unit test with Junit test.
- 5) Code example & explanation.
- 6) Review of the week.

1) GIT & GIT HUB (CONTINUE…)

2.1 What is Git ?

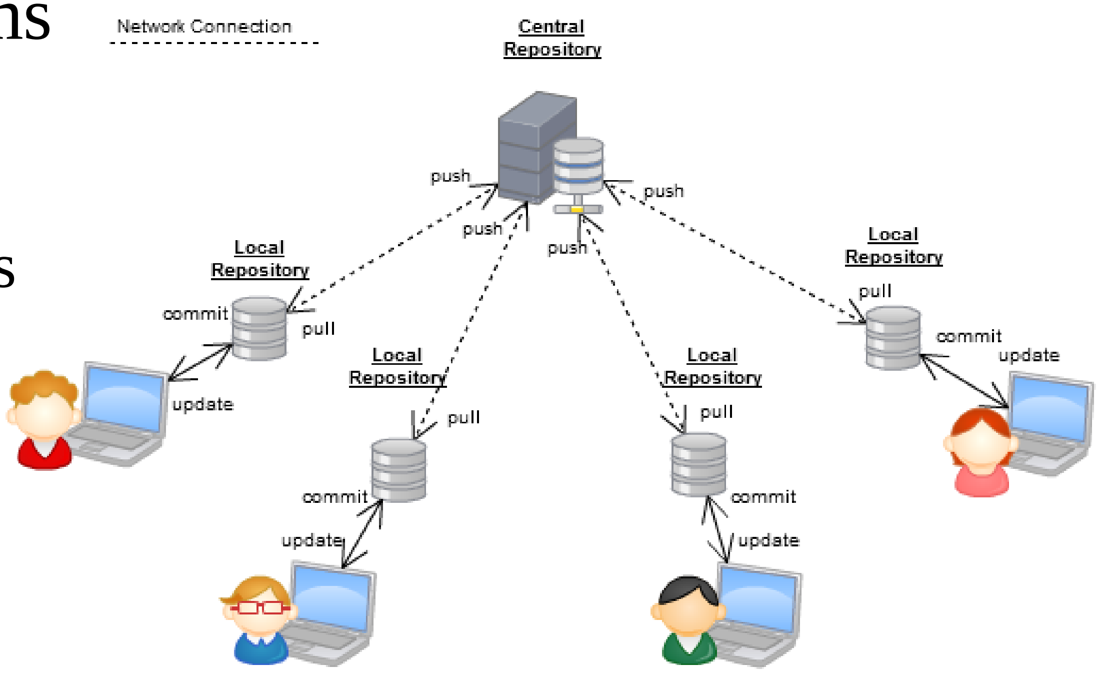
- Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

- As Git is a distributed version control system, it can be used as a server out of the box. Dedicated Git server software helps, amongst other features, to add access control, display the contents of a Git repository via the web, and help managing multiple repositories.

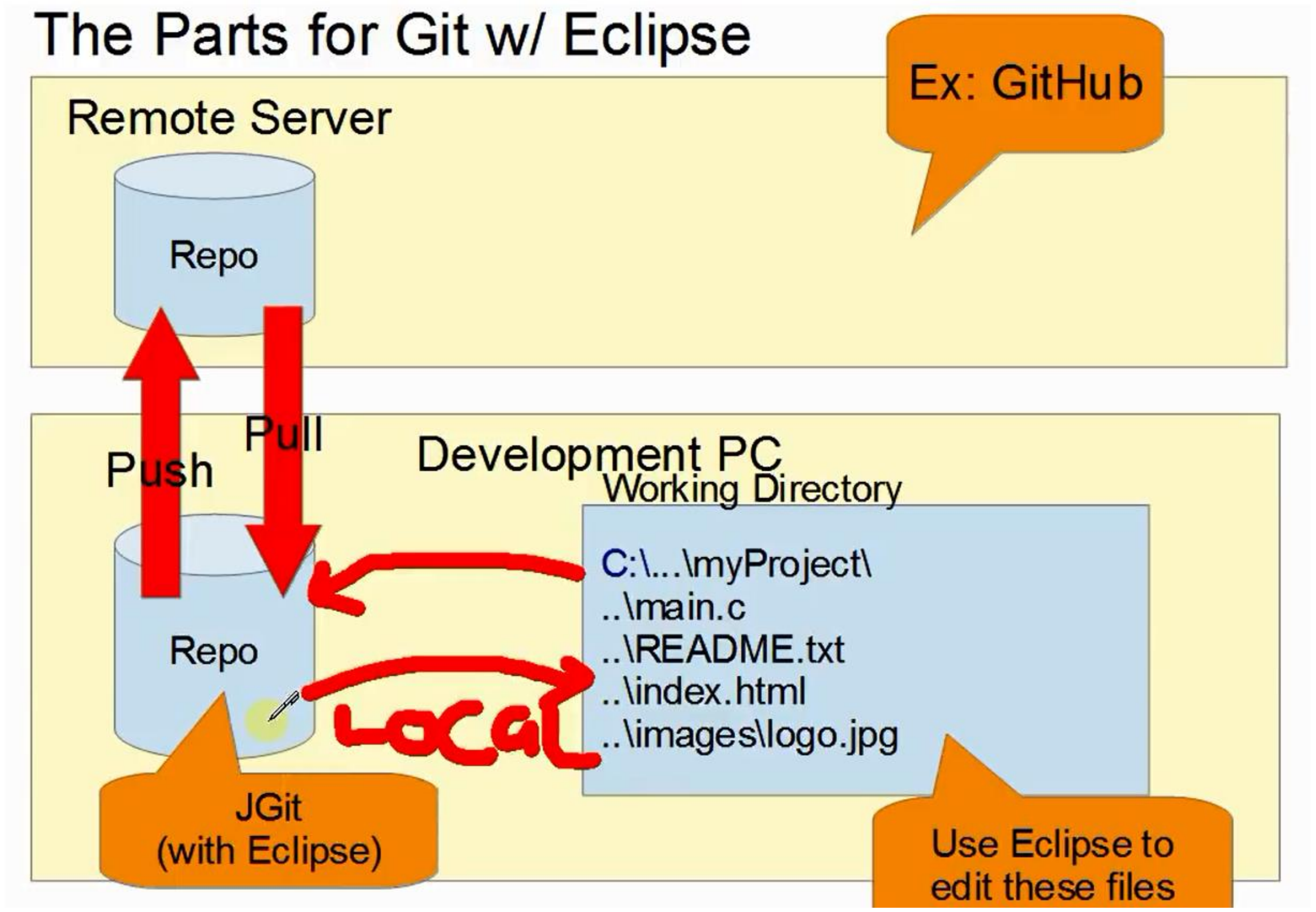


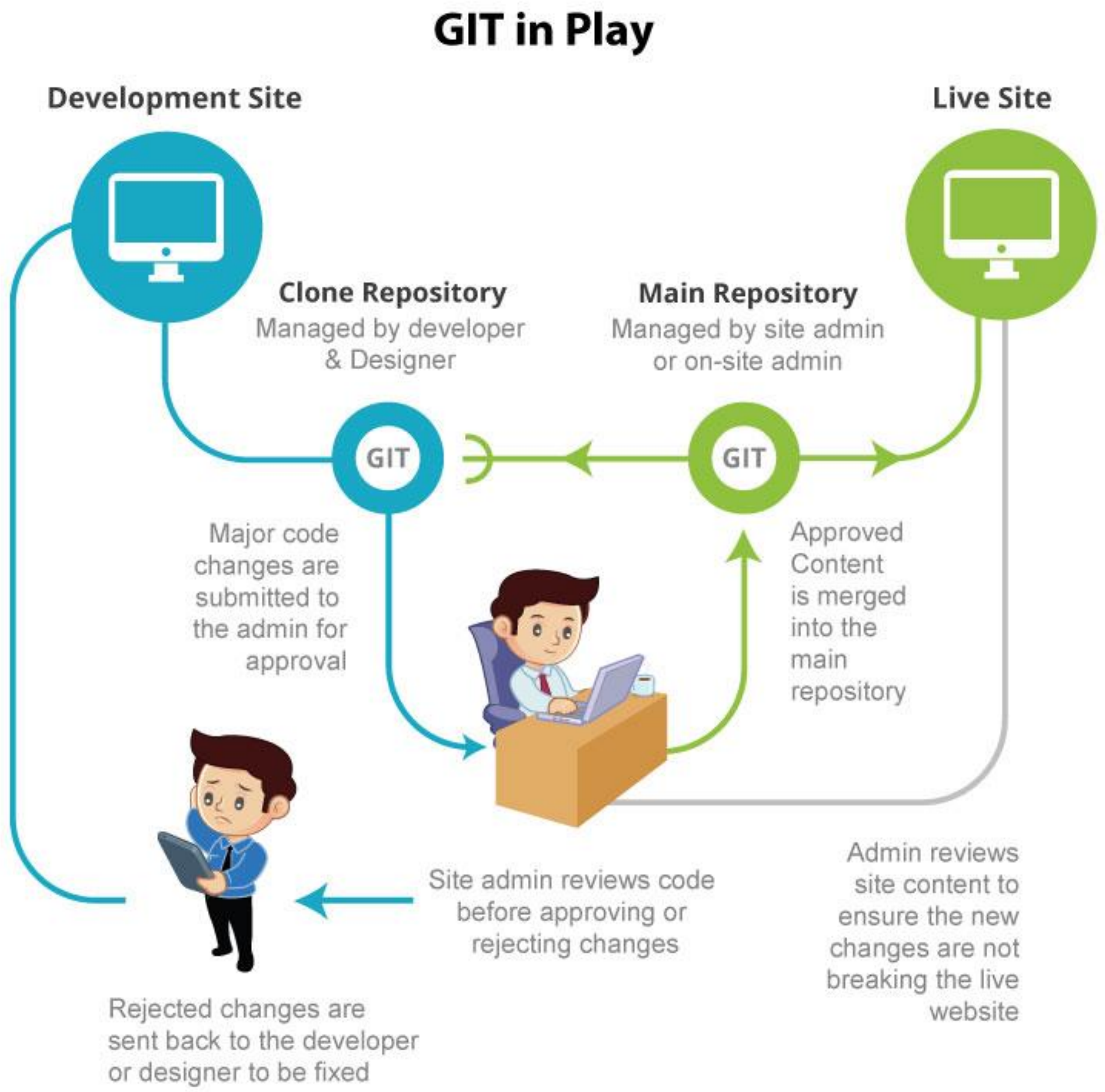
2.1 What is Git ?
(Why Git?)

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree
- Best competitor: Mercurial
  - I like Mercurial better
  - Same concepts, slightly simpler to use
  - In my (very limited) experience, the Eclipse plugin is easier to install and use
  - Much less popular than Git



2.1 What is Git ?
(EGit Environment)




GIT in Play
Development Site
Live Site
Clone Repository
Managed by developer & Designer
Main Repository
Managed by site admin or on-site admin
GIT
GIT
Major code changes are submitted to the admin for approval
Approved Content is merged into the main repository
Site admin reviews code before approving or rejecting changes
Admin reviews site content to ensure the new changes are not breaking the live website
Rejected changes are sent back to the developer or designer to be fixed

The basic Git workflow goes something like this:

You modify files in your working tree.

You stage the files, adding snapshots of them to your staging area.

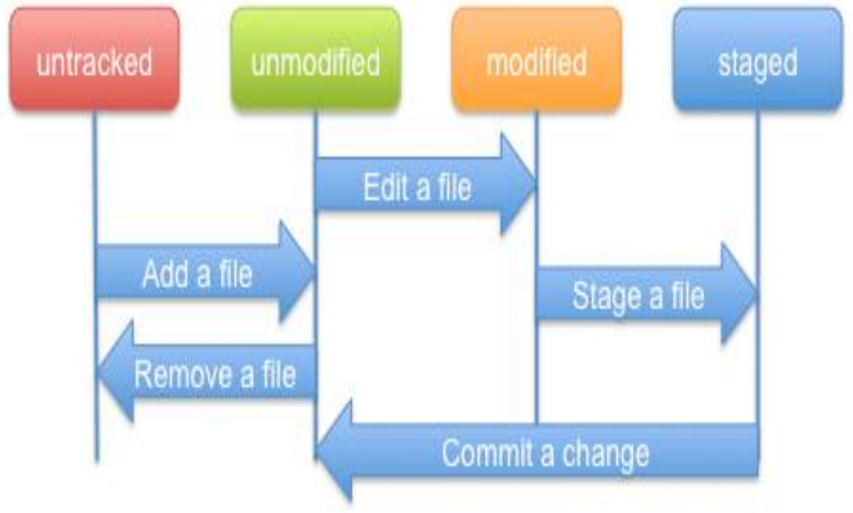
You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the Git directory, it’s considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

2.1 What is Git ?
(Some basic concepts...)

Recording Changes in the Repository

- You start from a fresh checkout of a branch of a local repository. You want to do some changes and record snapshots of these changes in the repository whenever you reach a state you want to record.
- Each file in the working directory can either be tracked or untracked:
- Tracked files are those which were in the last snapshot or files which have been newly staged into the index. They can be unmodified, modified, or staged.
- Untracked files are all other files (they were not in the last snapshot and have not yet been added to the index).
- When you first clone a repository, all files in the working directory will be tracked and unmodified since they have been freshly checked out and you haven't started editing them yet.
- As you edit files, git will recognize they are modified with respect to the last commit. You stage the modified files into the index and then commit the staged changes. The cycle can then repeat.



Download and Install Git

- There are online materials that are better than any that I could provide
- Here's the standard one: http://git-scm.com/downloads
- Here's one from StackExchange: http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

- Note: Git is primarily a command-line tool
- I prefer GUIs over command-line tools, but…
- The GIT GUIs are more trouble than they are worth (YMMV)

2.2 USING EGIT IN ECLIPSE

(INSTALLING EGIT IN ECLIPSE)

Eclipse install Git plugin – Step by Step installation instruction

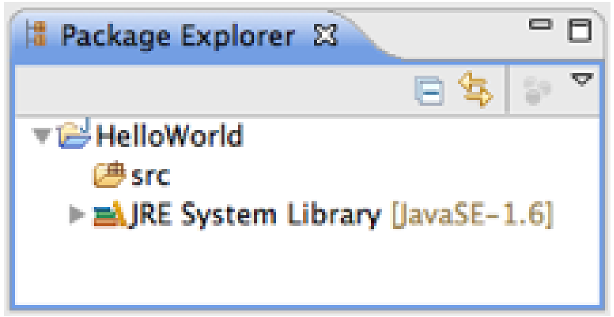
- Step 1) Go to: http://eclipse.org/egit/download/ to get the plugin repository location.
- Step 2.) Select appropriate repository location. In my case its http://download.eclipse.org/egit/updates
- Step 3.) Go to Help > Install New Software
- Step 4.) To add repository location, Click Add. Enter repository name as "EGit Plugin". Location will be the URL copied from Step 2. Now click Ok to add repository location.
- Step 5.) Wait for a while and it will display list of available products to be installed. Expend "Eclipse Git Team   Provider" and select "Eclipse Git Team Provider". Now Click Next
- Step 6.) Review product and click Next.
- Step 7.) It will ask you to Accept the agreement. Accept the agreement and click Finish.
- Step 8.) Within few seconds, depending on your internet speed, all the necessary dependencies and executable will be downloaded and installed.
- Step 9.) Accept the prompt to restart the Eclipse.
- Your Eclipse Git Plugin installation is complete.

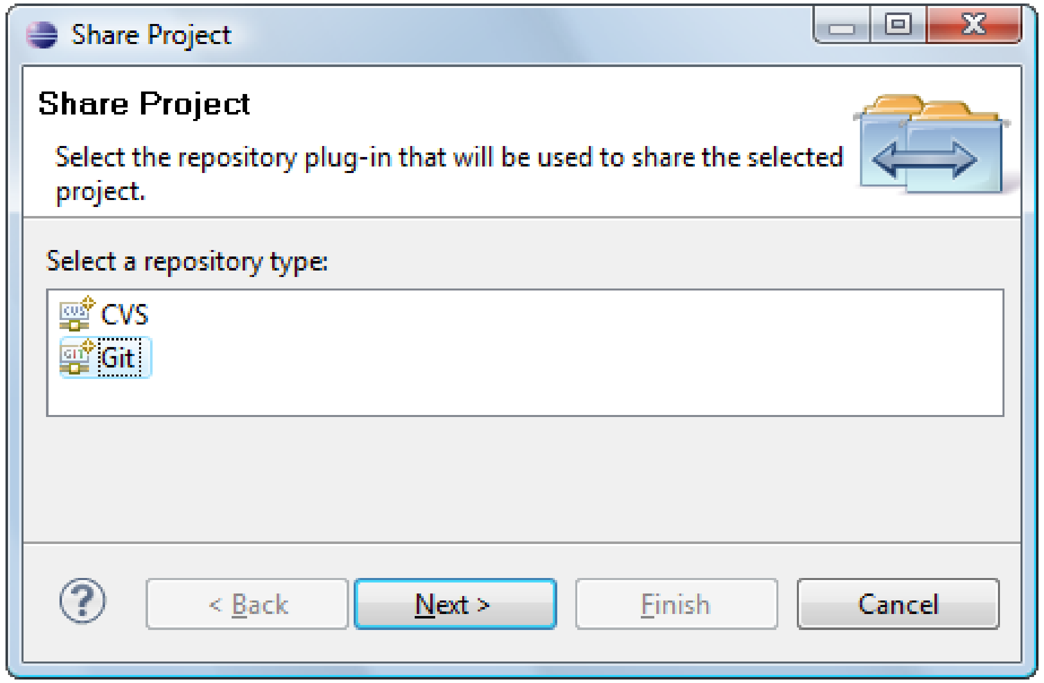
To verify your installation.

- Step 1.) Go to Help > Install New Software
- Step 2.) Click on Already Installed and verify plugin is installed.

2.2 Using EGit in Eclipse
(Creating Repository)

- Create a new Java project, eg. HelloWorld.
- Select the project, click **File > Team > Share Project**.
- Select repository type **Git** and click **Next**.




Share Project

Share Project

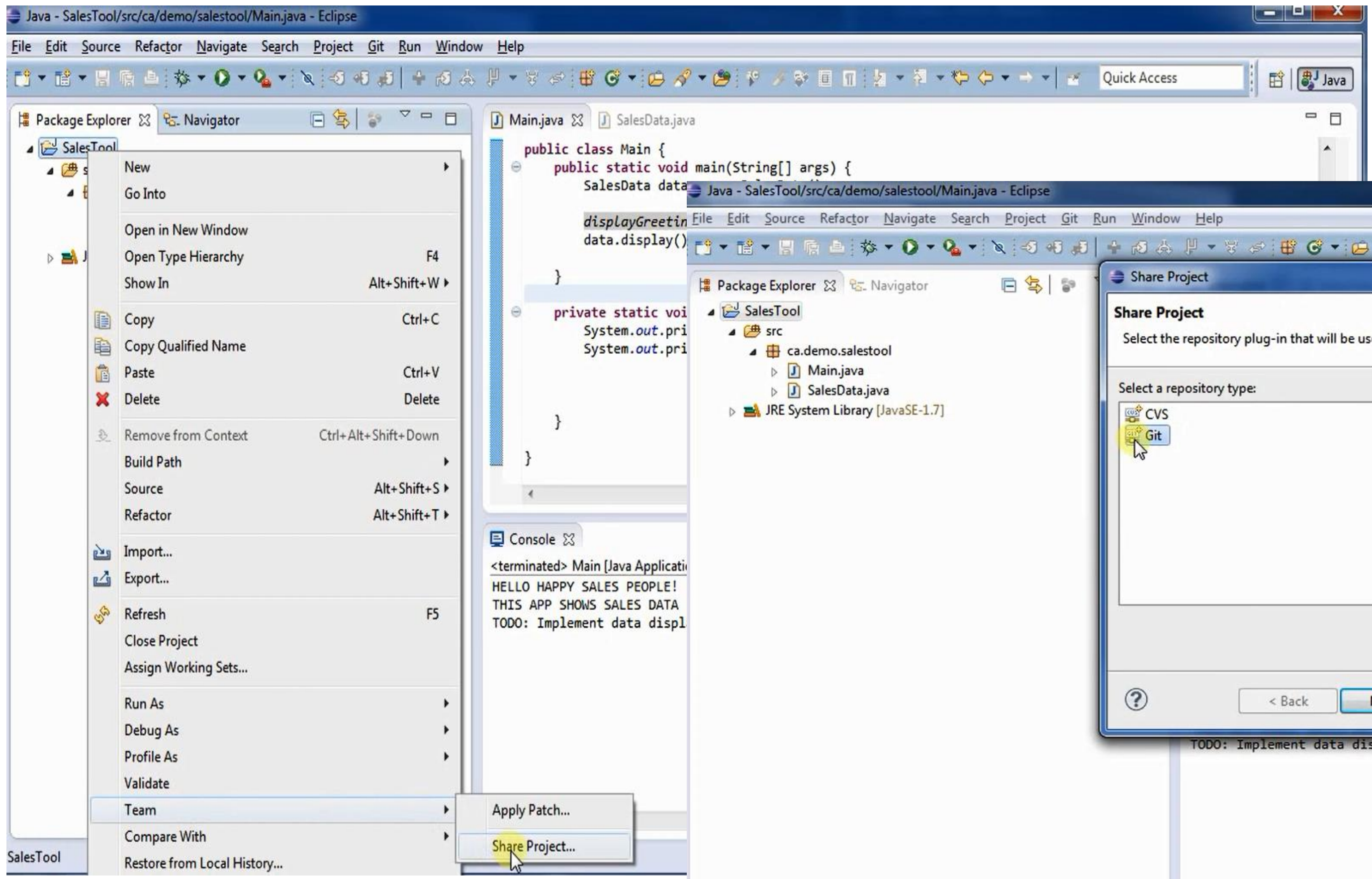
Select the repository plug-in that will be used to share the selected project.

Select a repository type:

CVS
Git

2.2 USING EGIT IN ECLIPSE

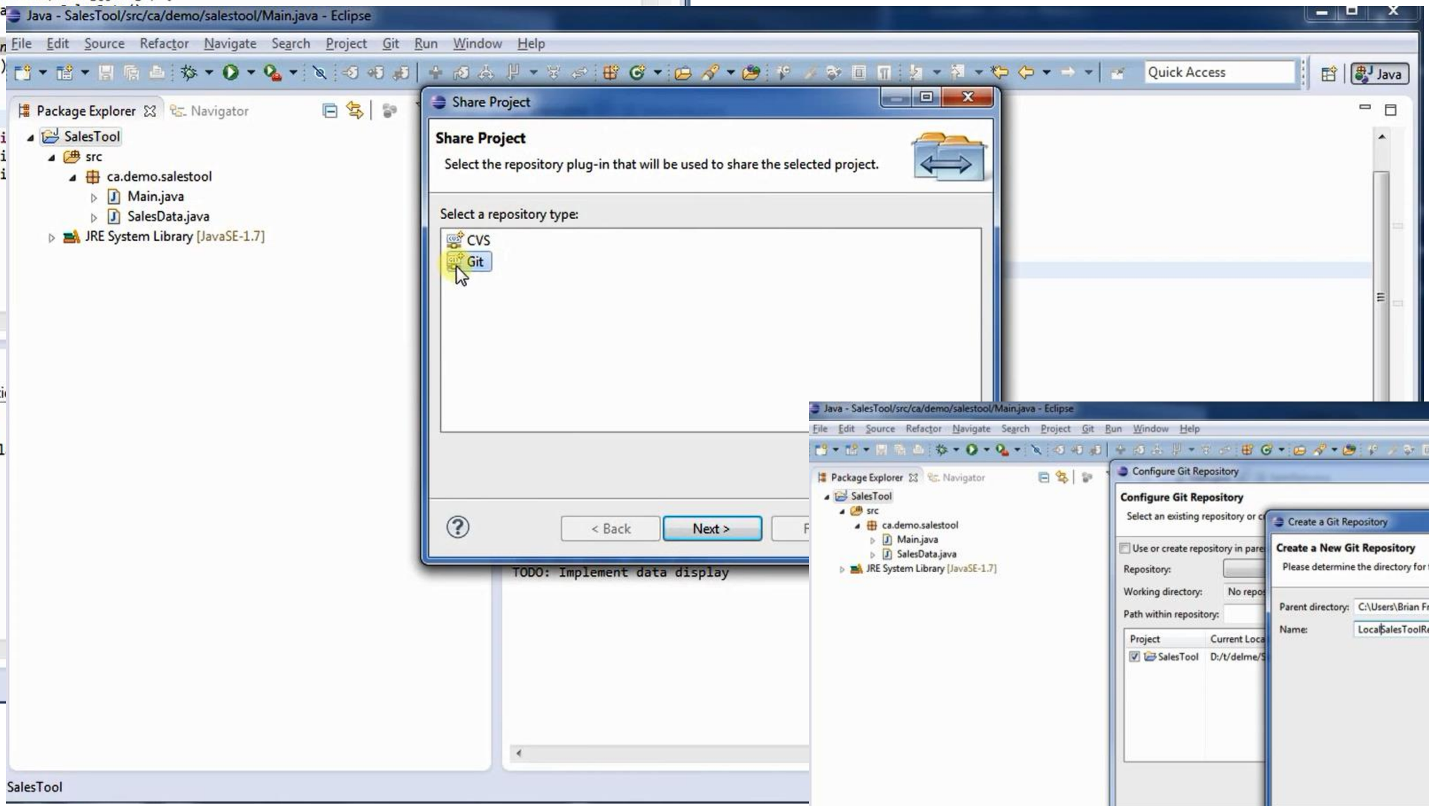
(CREATING REPOSITORY)


Java - SalesTool/src/ca/demo/salestool/Main.java - Eclipse

```
public class Main {
    public static void main(String[] args) {
        SalesData data
        
        displayGreetin
        data.display()
    }
    
    private static voi
        System.out.pri
        System.out.pri
        
        
    }
}
```

Console

```
<terminated> Main [Java Applicati
HELLO HAPPY SALES PEOPLE!
THIS APP SHOWS SALES DATA
TODO: Implement data displ
```

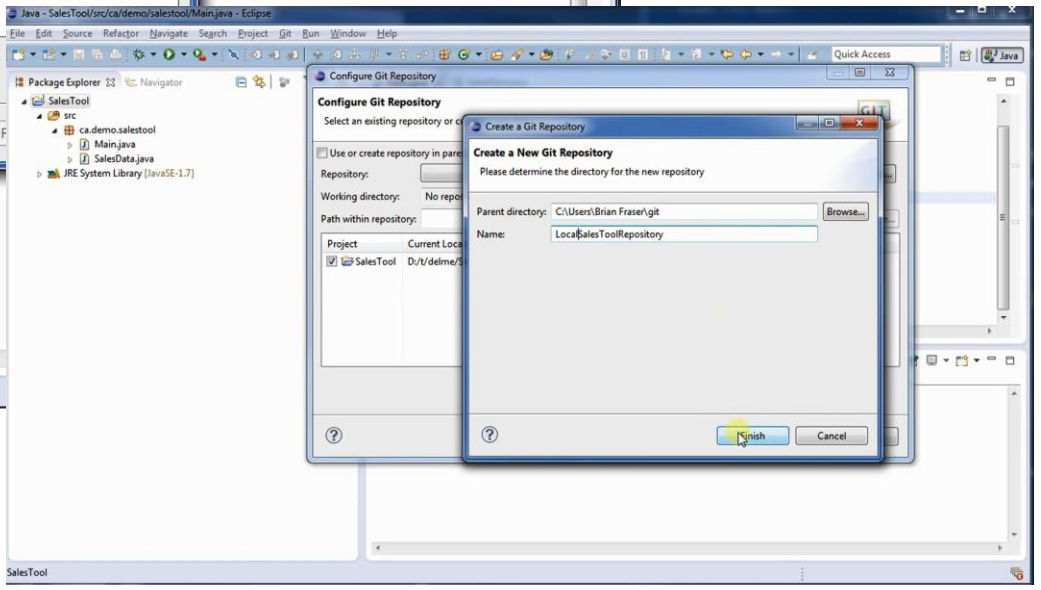

Java - SalesTool/src/ca/demo/salestool/Main.java - Eclipse

Share Project

Select the repository plug-in that will be used to share the selected project.

Select a repository type:

```
TODO: Implement data display
```


Java - SalesTool/src/ca/demo/salestool/Main.java - Eclipse

Configure Git Repository

Select an existing repository or c

Use or create repository in pare

Repository:

Working directory: No repos

Path within repository:

Project	Current Loca
✓ SalesTool	D:/t/delme/S

Create a Git Repository

Create a New Git Repository

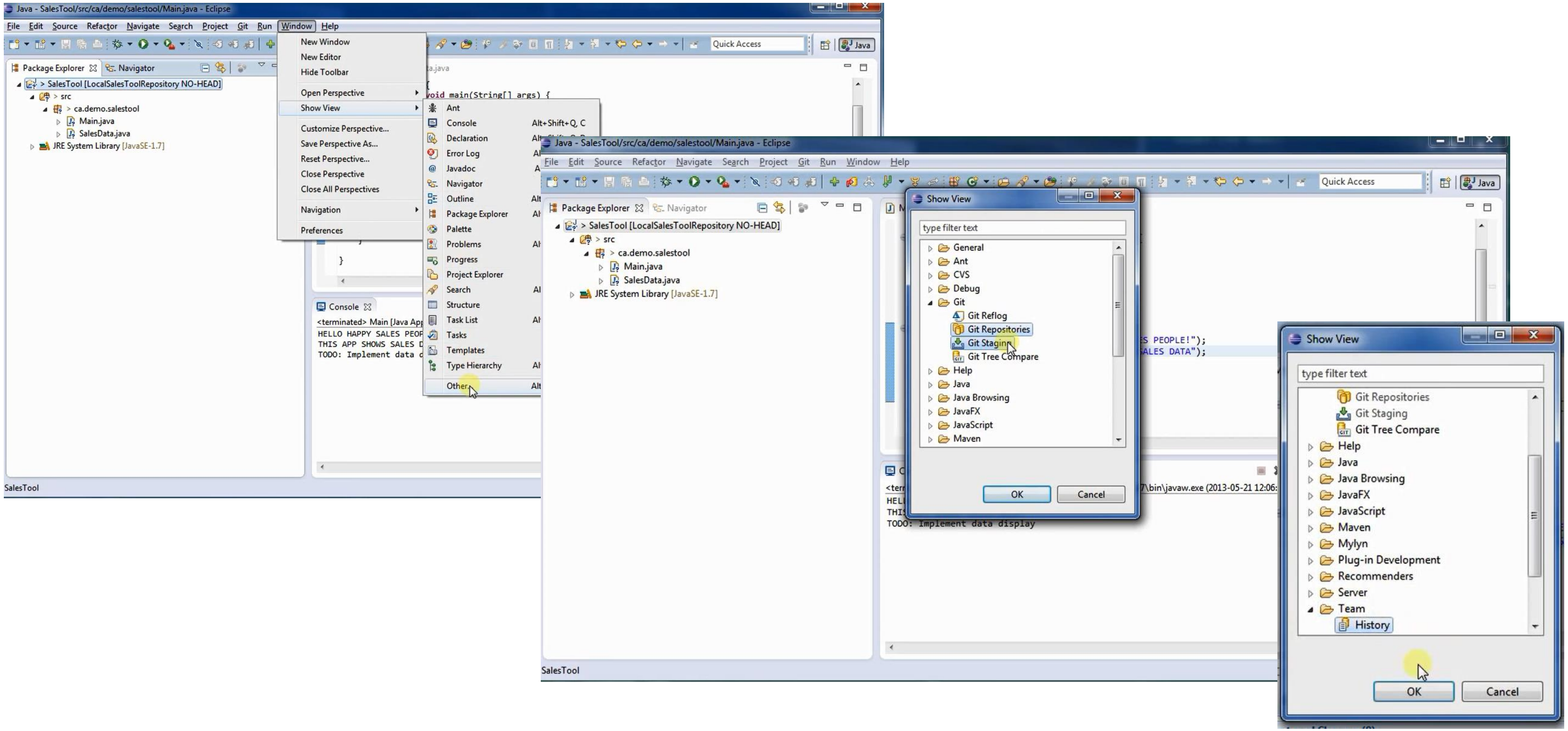
Please determine the directory for the new repository

Parent directory: C:\Users\Brian Fraser\git

Name: LocalSalesToolRepository

2.2 USING EGIT IN ECLIPSE

(CREATING REPOSITORY)



```
void main(String[] args) {
```

```
<terminated> Main [Java App
HELLO HAPPY SALES PEOP
THIS APP SHOWS SALES D
TODO: Implement data d
```

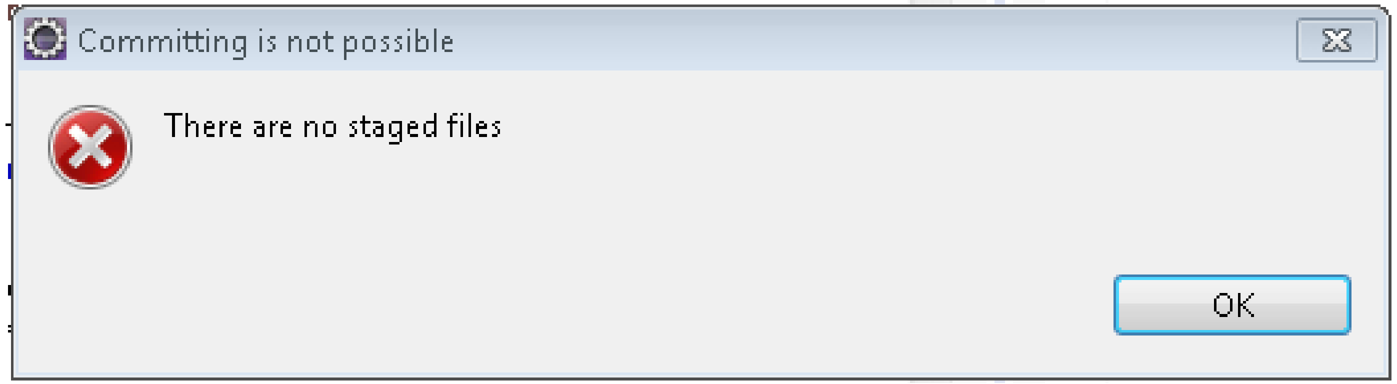
```
S PEOPLE!");
ALES DATA");
```

```
7\bin\javaw.exe (2013-05-21 12:06:
TODO: Implement data display
```

2.2 Using EGit in Eclipse

(Creating Repository)

- Error: Git commit not possible. There are no staged files



Committing is not possible

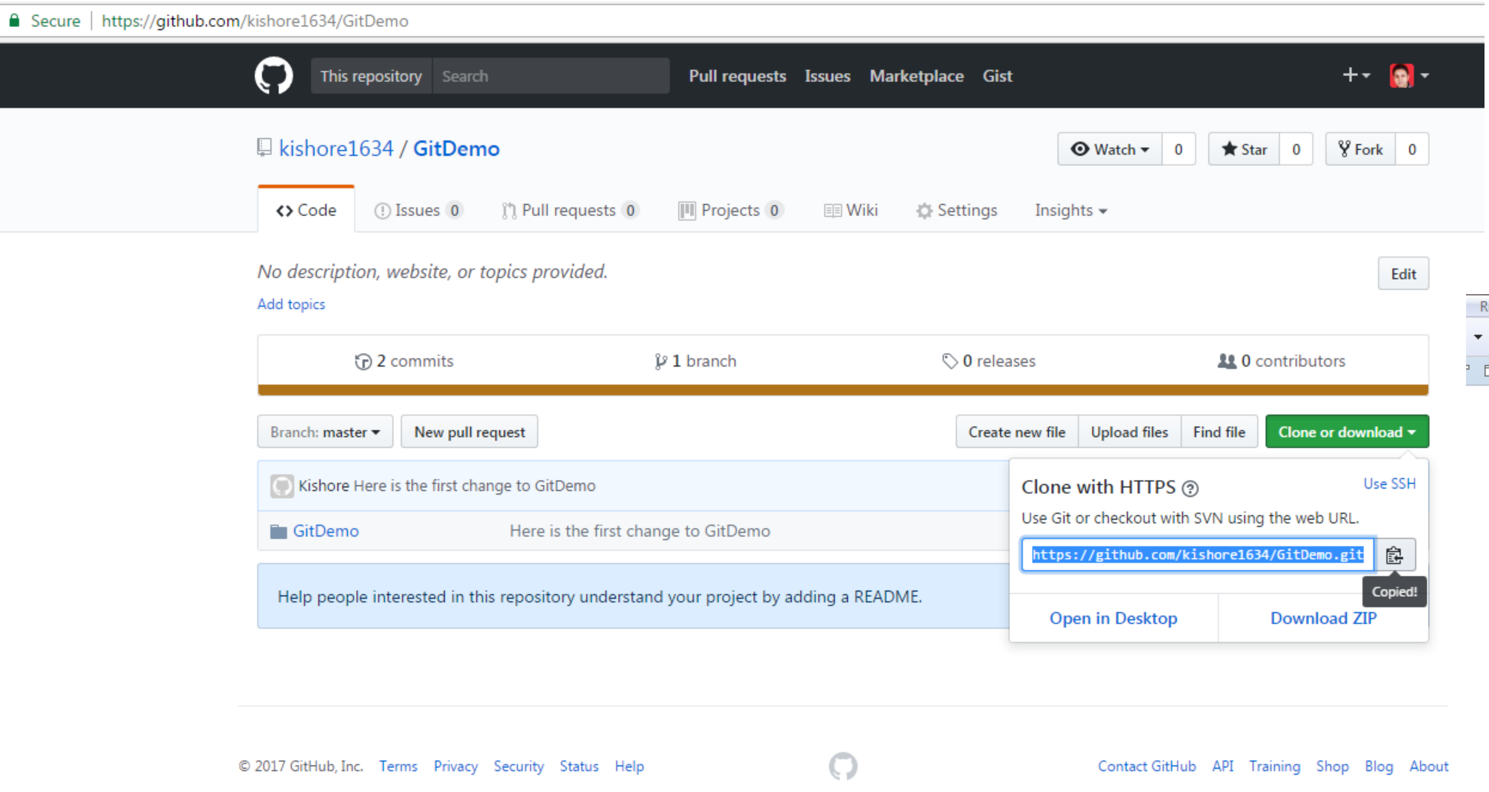
There are no staged files

Solution: using EGit: Right click it and navigate to Team => Add.

after then Commit & Push the code

2.2 USING EGIT IN ECLIPSE

(DOWNLOADING REPOSITORY FROM A GIT LINK)


Secure | https://github.com/kishore1634/GitDemo

kishore1634 / GitDemo

No description, website, or topics provided.

2 commits | 1 branch | 0 releases | 0 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

Kishore Here is the first change to GitDemo

GitDemo — Here is the first change to GitDemo

Help people interested in this repository understand your project by adding a README.

Clone with HTTPS — Use SSH

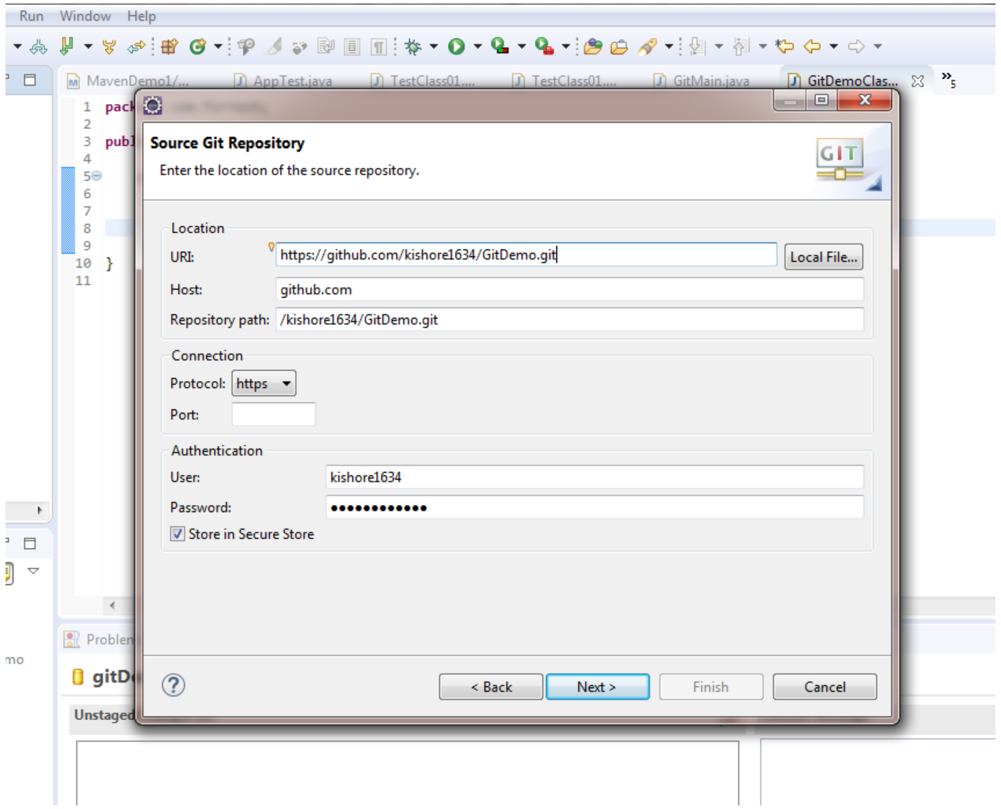
Use Git or checkout with SVN using the web URL.

```
https://github.com/kishore1634/GitDemo.git
```

Copied!

Open in Desktop | Download ZIP

© 2017 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub API Training Shop Blog About


Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/kishore1634/GitDemo.git

Host: github.com

Repository path: /kishore1634/GitDemo.git

Connection

Protocol: https

Port:

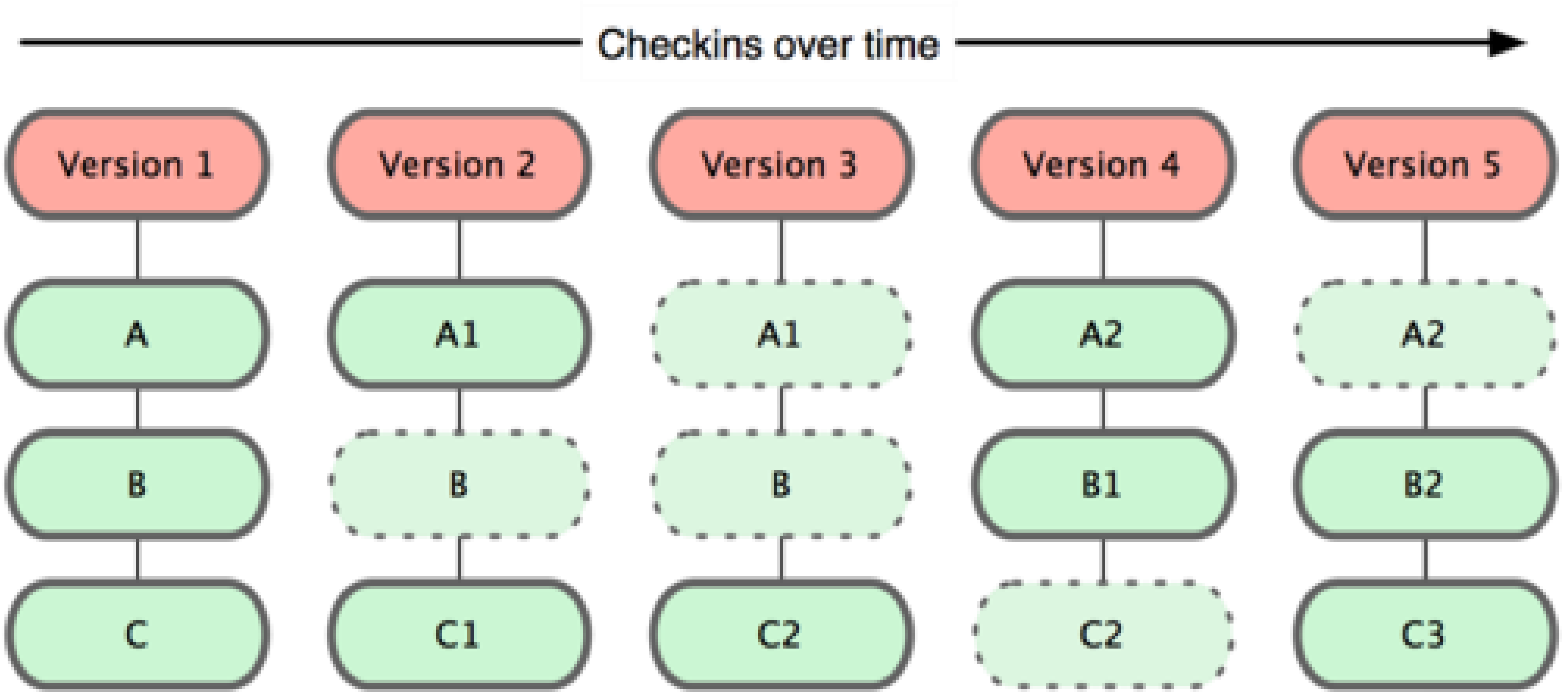
Authentication

User: kishore1634

Password: ••••••••••••

[✓] Store in Secure Store

GIT AND VERSIONS



Commits and graphs

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no “parents”
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent
  - You can also merge two commit objects to form a new one
    - The new commit object has two parents
- Hence, commit objects form a **directed graph**
  - Git is all about using and manipulating this graph

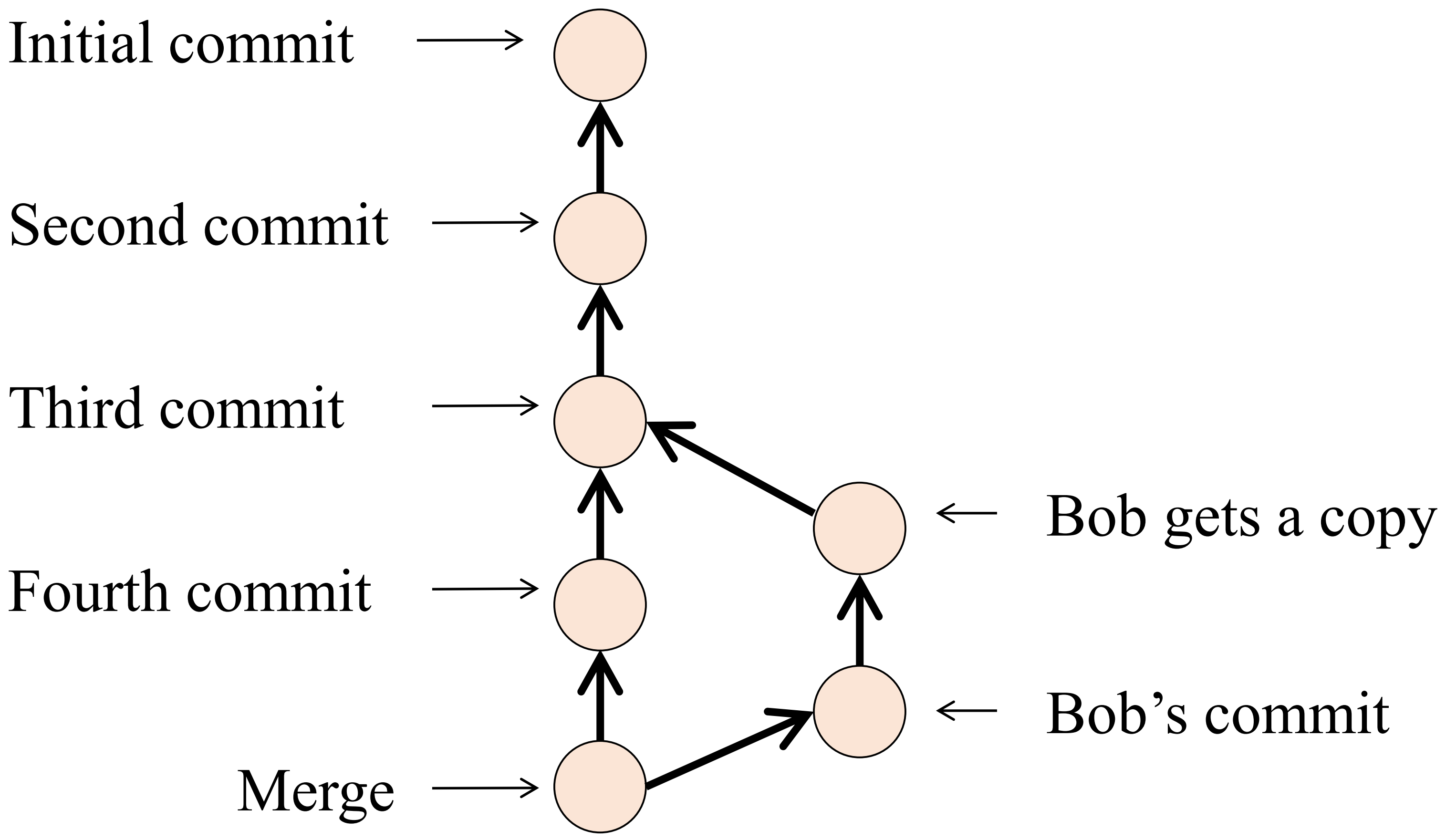
Working with your own repository

- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
  - This results in a linear graph: A → B → C → …→ HEAD
- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph

Commit Messages

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

Multiple Versions



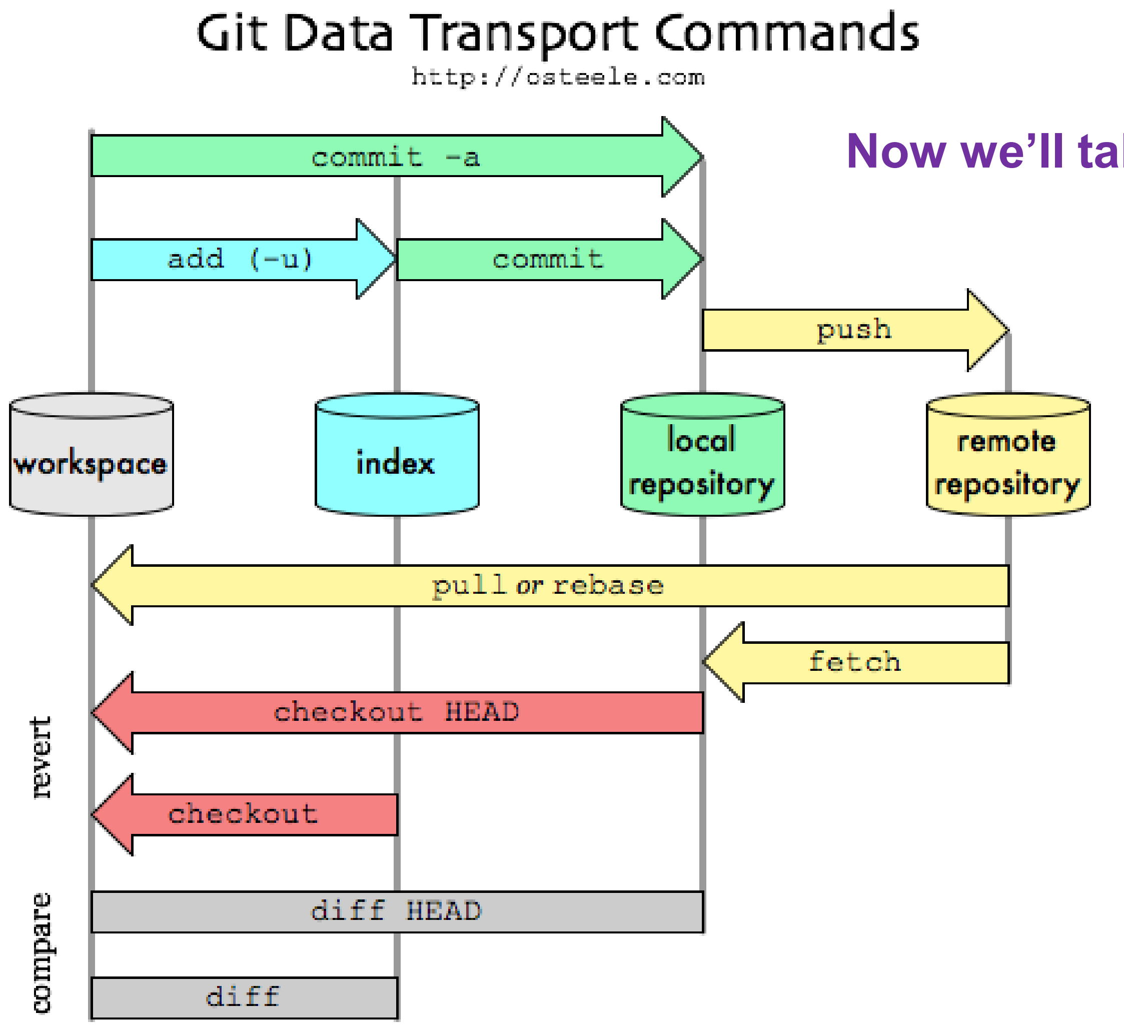
Keeping it Simple

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don’t have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these

- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently

- If you need help: https://help.github.com/

The Big Picture



Now we'll talk about this part

GITHUB

- Largest open source git hosting site
- Public and private options
  - Public: everyone can see, but must be a collaborator to modify files
  - Public repos often used for large open source or to distribute libraries etc.
  - Private: only those you choose as collaborators can see/update
- Try it
  - Log onto github
  - Walk through the tutorials

Getting an Account

- Set up a user account
  - Public account is free
  - Remember your password!
- Get an educational account
  - Students get 5 private repos
  - After your account is created, send a discount request (may take a week or so to get a response)
  - Discount button on: https://education.github.com/
- CS collaboration policy
  - Your homework must be stored in a *private* repository.

Creating a Repo

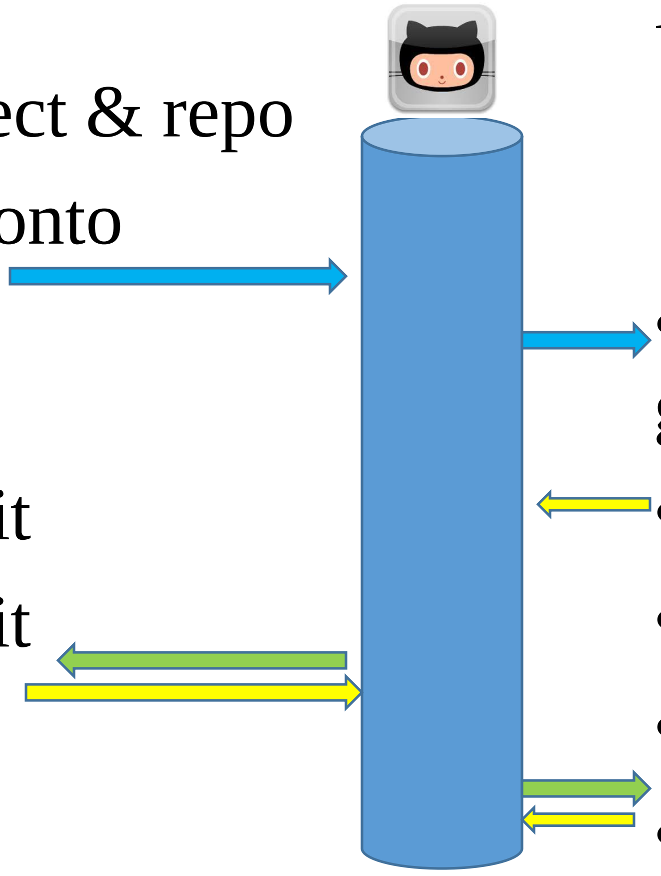
Option 1

- Create repo on github
- Add collaborators
- Clone onto all machines

Option 2

- Use existing repo
- Create repo on github
- Add collaborators
- Set up the connection between existing repo and github repo (called a *remote*)
- Push code onto github
- Clone onto other machines

Typical Workflow (Option 2)



Person A

- Setup project & repo
- push code onto github
- edit/commit
- edit/commit
- pull/push

Person B

- clone code from github
- edit/commit/push
- edit…
- edit… commit
- pull/push

This is just the flow, specific commands on following slides.
It's also possible to create your project first on github, then clone (i.e., no git init)

Remote Repository

- Remote repository is a *bare repository*
- There is no working directory
- Four transfer protocols
  - http – this is what I recommend/use
  - local (not covered – good for shared filesystems)
  - git (not covered – fast but more setup)
  - SSH (used to be most common)
- Use https on the client:

  git clone https://github.com/CyndiRader/JavaDemos.git

protocol **userID** **repository**

When you create a repo on github, it will show you the HTTPS clone URL
Avoid typos: a) copy the URL, b) in git bash, click git-icon/edit/paste OR press Insert.

CREATE THE REPO – OPTION 1

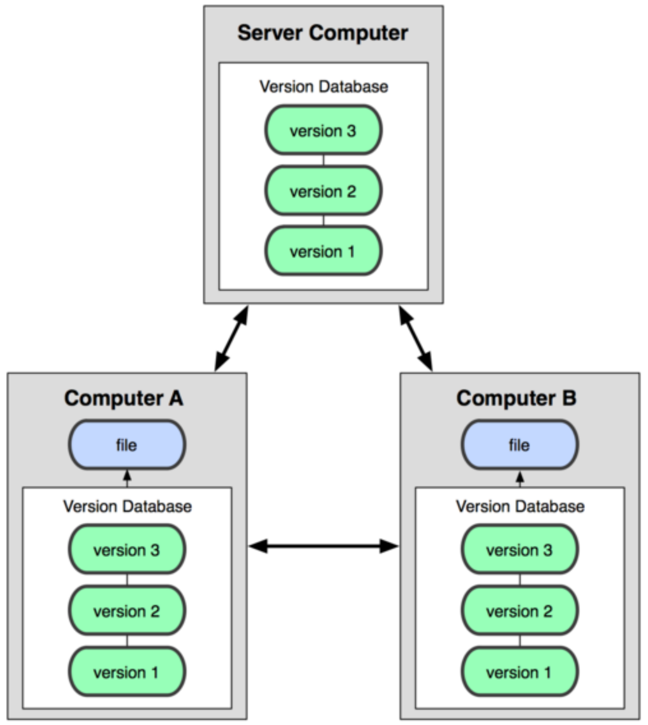
- Log onto github
- Click on + to add repository
- Enter name
- Add .gitignore, good to have a README
- Click on Settings to control access (Collaborators tab)
- On PC, do git clone

Create the Repo – Option 2

- Create your Java Project
- Create your local repo
  - git init
  - git commit –m “Initial commit”
- Create a repo on github
- Add a “shortname” for your git repository
  - git remote add [shortname] [url]
  - git remote add origin https://github.com:[user name]/[repository name].git
  - Ex: git remote add origin https://github.com:CyndiRader/JavaDemos.git
  - Remember: You can copy/paste the repo url from github
- Push your code onto github
  - git push –u [remote-name] [branch-name].
  - Ex: git push –u origin master – enter username/password

COLLABORATING VIA GITHUB - CLONING

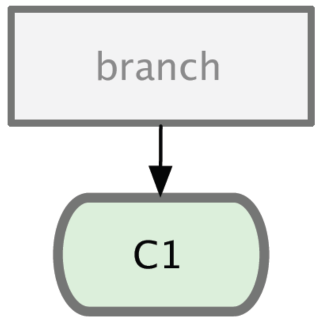
- git clone adds the remote repository under the name origin
- git clone https://github.com:[user name]/[repository name].git
    - git clone https://github.com:CyndiRader/JavaDemos.git

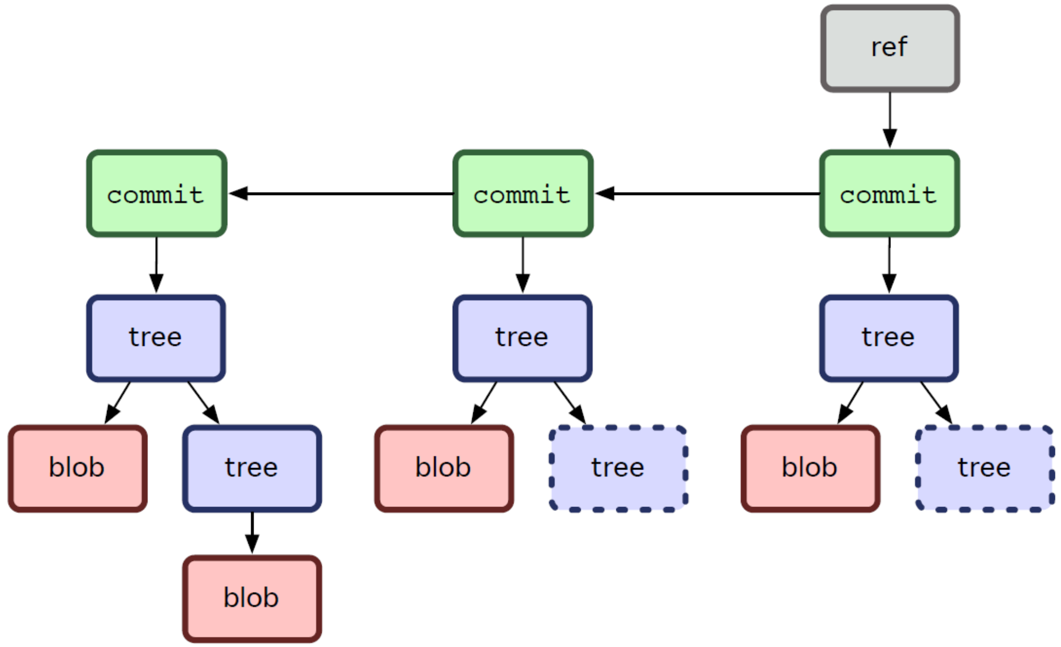


Branching

- Git sees commit this way…
- Branch annotates which commit we are working on

lightweight, movable pointers to a commit





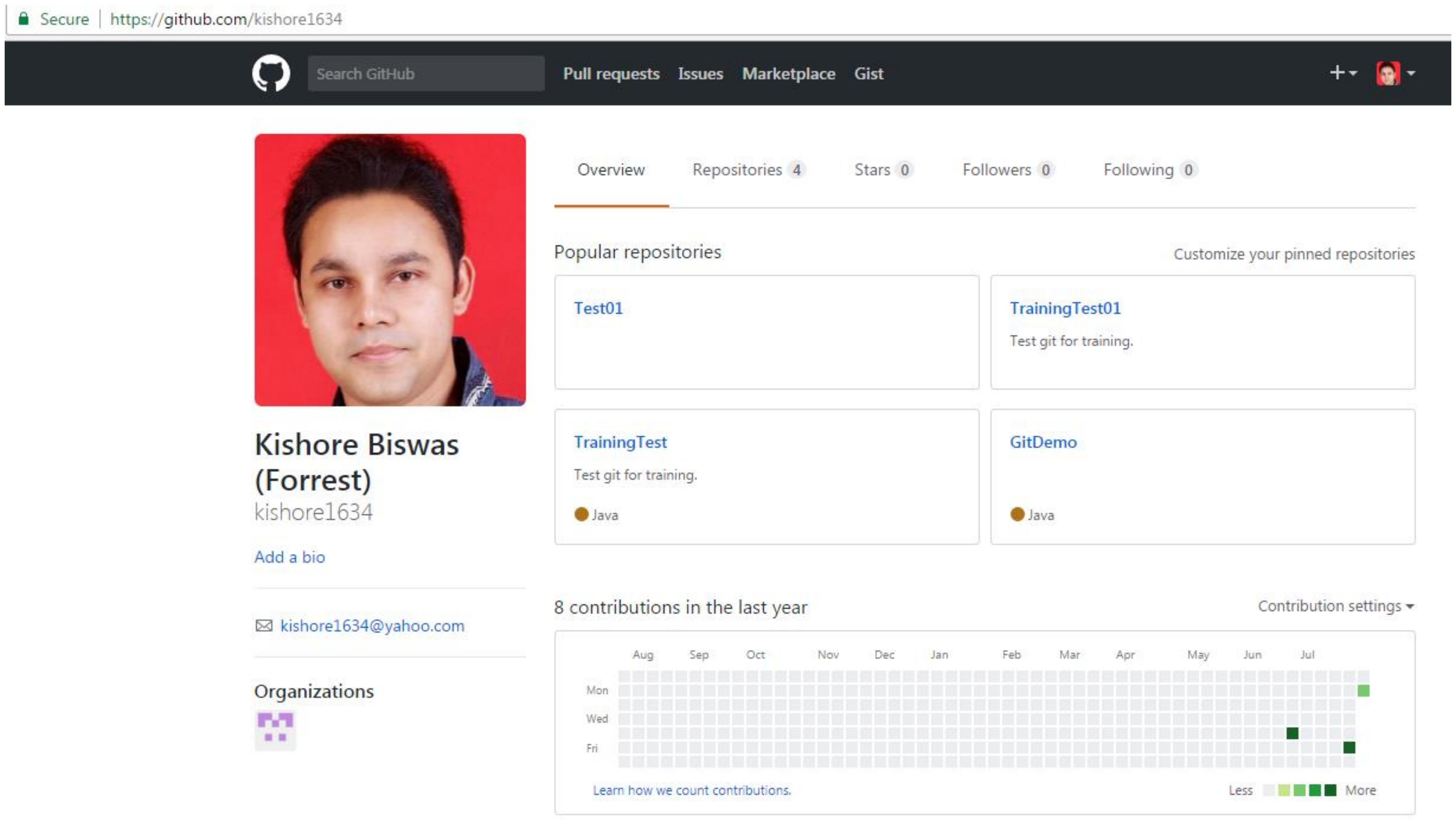
Branching and Merging

- Why this is cool?
  - Non-linear development

clone the code that is in production
create a branch for issue #53 (iss53)
work for 10 minutes
someone asks for a hotfix for issue #102
checkout ‘production’
create a branch (iss102)
fix the issue
checkout ‘production’, merge ‘iss102’
push ‘production’
checkout ‘iss53’ and keep working

WORKING WITH REMOTE…

- Use git remote add to add an remote repository


Secure | https://github.com/kishore1634

Search GitHub Pull requests Issues Marketplace Gist

Overview Repositories 4 Stars 0 Followers 0 Following 0

Kishore Biswas (Forrest)
kishore1634

Add a bio

kishore1634@yahoo.com

Organizations

Popular repositories — Customize your pinned repositories

Test01

TrainingTest01
Test git for training.

TrainingTest
Test git for training.
Java

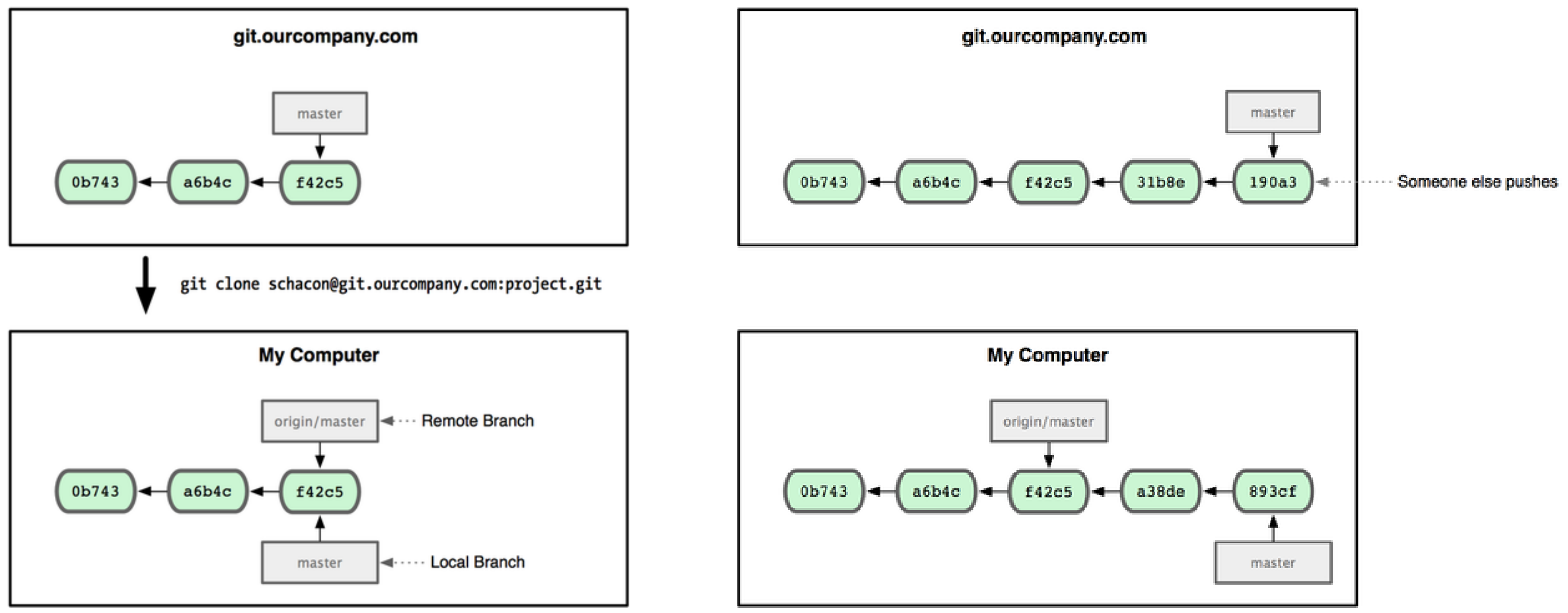
GitDemo
Java

8 contributions in the last year — Contribution settings

Learn how we count contributions.

Working with remote…

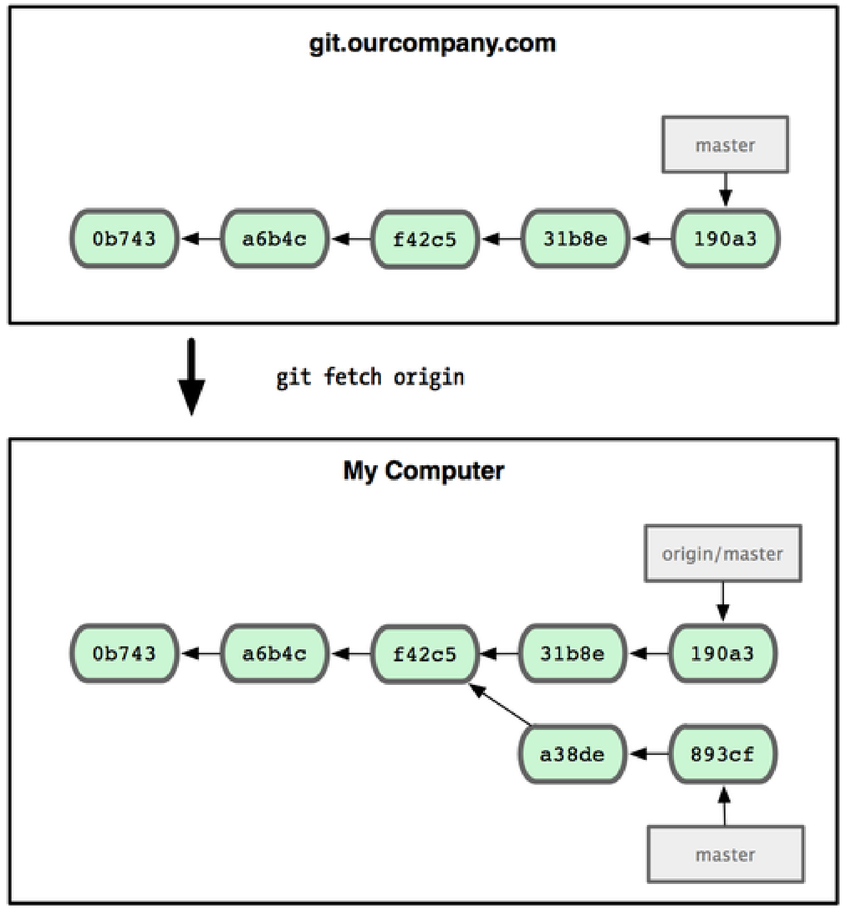
- Remote branching
  - Branch on remote are different from local branch



```
git clone schacon@git.ourcompany.com:project.git
```

WORKING WITH REMOTE…

- Remote branching
  - Branch on remote are different from local branch
  - Git fetch origin to get remote changes
  - Git pull origin try to fetch reomte changes and merge it onto current branch



Git push remote_name branch_name

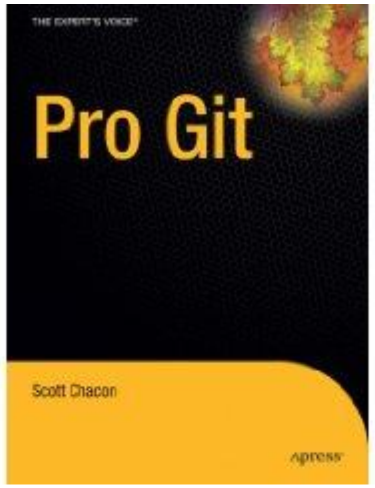
Share your work done on branch_name to remote remote_name

Summary…

- We covered fundamentals of Git
  - Three trees of git
    - HEAD, INDEX and working directory
  - Basic work flow
    - Modify, stage and commit cycle
  - Branching and merging
    - Branch and merge
  - Remote
    - Add remote, push, pull, fetch
  - Other commands
    - Revert change, history view

SUMMARY

- However, this is by no means a complete portray of git, some advanced topics are skipped:
  - Rebasing
  - Commit amend
  - Distributed workflow
- For more information, consult
  - Official document
  - Pro Git
    - Free book available at http://progit.org/book/



2) UNIT TEST WITH JUNIT

INTRODUCTION

- Programmers have a nasty habit of not testing their code properly as they develop their software.

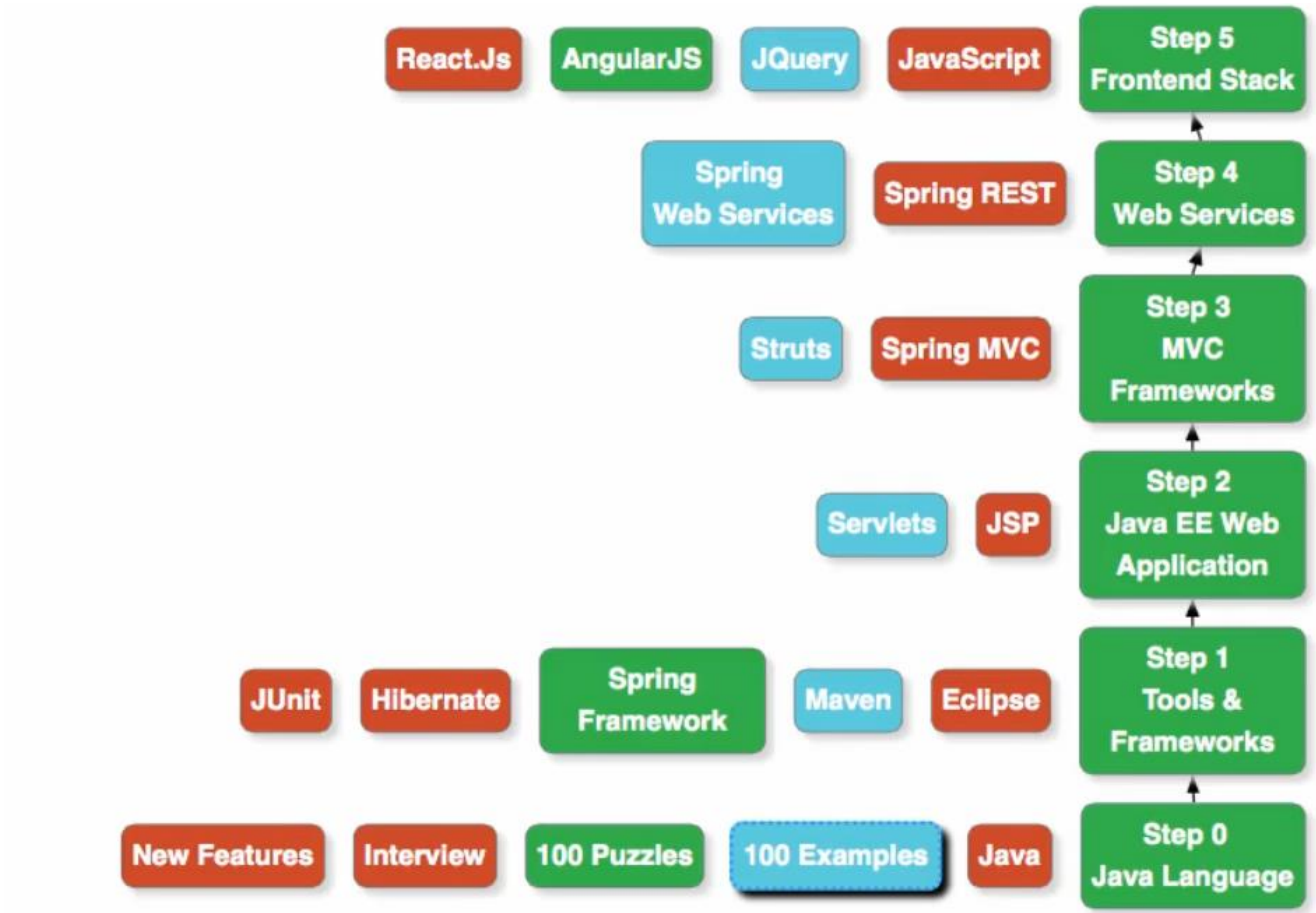
- This prevents you from measuring the progress of development- you can't tell when something starts working or when something stops working.

- Using *JUnit* you can cheaply and incrementally build a test suite that will help you measure your progress, spot unintended side effects, and focus your development efforts.

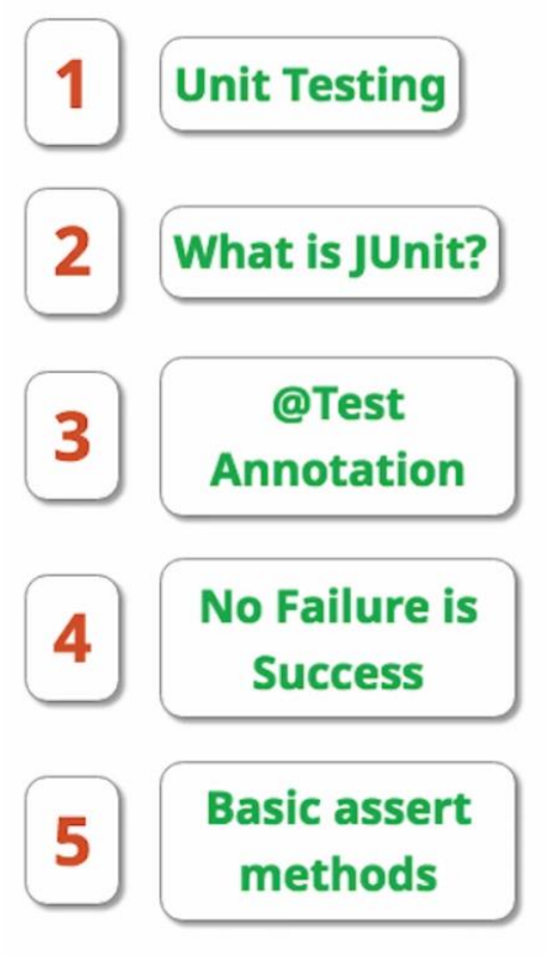
The Problem, (The Vicious Cycle)

- Every programmer knows they should write tests for their code. Few do. The universal response to "Why not?" is "I'm in too much of a hurry." This quickly becomes a vicious cycle- the more pressure you feel, the fewer tests you write. The fewer tests you write, the less productive you are and the less stable your code becomes. The less productive and accurate you are, the more pressure you feel.

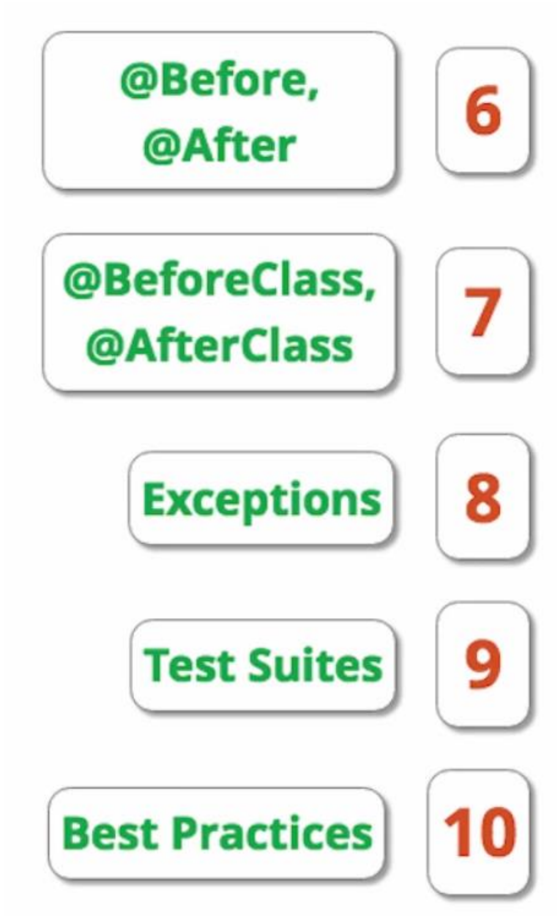
JAVA EE BASIC PATH ROUT.



UNIT TESTING PATH

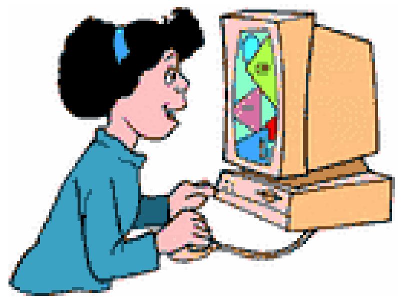


1. Unit Testing
2. What is JUnit?
3. @Test Annotation
4. No Failure is Success
5. Basic assert methods



6. @Before, @After
7. @BeforeClass, @AfterClass
8. Exceptions
9. Test Suites
10. Best Practices

EXAMPLE



- In this example pay attention to the interplay of the code and the tests.
- The style here is to write a few lines of code, then a test that should run, or even better, to write a test that won't run, and then write the code that will make it run.

Example: Currency Handler Program

- The program we write will solve the problem of representing arithmetic with multiple currencies.

- Things get more interesting once multiple currencies are involved. You cannot just convert one currency into another for doing arithmetic since there is no single conversion rate- you may need to compare the value of a portfolio at yesterday's rate and today's rate.

Money.class

- Let's start simple and define a class “Money.class” to represent a value in a single currency.
- We represent the amount by a *simple int*.
- We represent a currency as a string holding the ISO three letter abbreviation (USD, CHF, etc.).

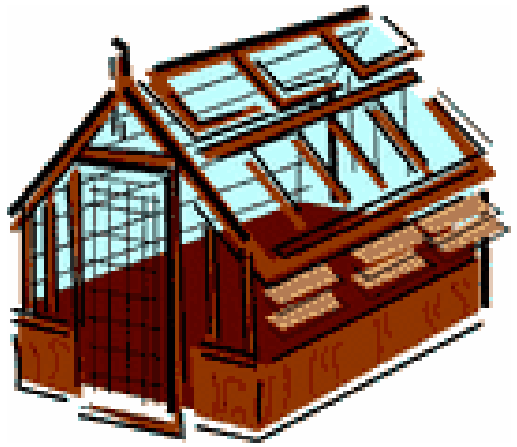
Money.class – Let's Code

```
class Money {
    private int fAmount;
    private String fCurrency;
    public Money(int amount, String currency) {
        fAmount= amount;
        fCurrency= currency;
    }

    public int amount() {
        return fAmount;
    }

    public String currency() {
        return fCurrency;
    }

    public Money add(Money m) {
        return new Money(amount()+m.amount(), currency());
    }
}
```

When you add two Moneys of the same currency, the resulting Money has as its amount the sum of the other two amounts.

LET'S START TESTING

- Now, instead of just coding on, we want to get immediate feedback and practice "code a little, test a little, code a little, test a little".
- To implement our tests we use the *JUnit framework*.



JUNIT CLASSES

- JUnit classes are important classes, used in writing and testing JUnits. Some of the important classes are –

- Assert – Contains a set of assert methods.
- TestCase – Contains a test case that defines the fixture to run multiple tests.
- TestResult – Contains methods to collect the results of executing a test case.

TESTING MONEY.CLASS

- JUnit defines how to structure your test cases and provides the tools to run them. You implement a test in a subclass of TestCase. To test our Money implementation we therefore define MoneyTest.class as a subclass of TestCase. We add a test method testSimpleAdd, that will exercise the simple version of Money.add() above. A JUnit test method is an ordinary method without any parameters.

Testing Money.class (MoneyTest.class)

```
import junit.framework.*;

public class MoneyTest extends TestCase {
    //…
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");  // (1)
        Money m14CHF= new Money(14, "CHF");
        Money expected= new Money(26, "CHF");
        Money result= m12CHF.add(m14CHF);    // (2)
        Assert.assertTrue(expected.equals(result));  // (3)
        Assert.assertEquals(m12CHF, new Money(12, "CHF")); // (4)
    }
}
```

(2) Code which exercises the objects in the fixture.

(3) Code which verifies the results. If not true, JUnit will return an error.

(1) Code which creates the objects we will interact with during the test. This testing context is commonly referred to as a test's *fixture*. All we need for the testSimpleAdd test are some Money objects.

(4) Since assertions for equality are very common, there is also an Assert.assertEquals convenience method. If not equal JUnit will return an error.

Assert Function

- In the new release of JUnit the Assert function is simplified. Instead of writing Assert.assertTrue(…); you can simply write assertTrue(…);.

Assert Functions

- assertTrue() – Returns error if not true.
- assertFalse() – Returns error if not false.
- assertEquals() – Returns error if not equal.
- assertNotSame(Object, Object) – Returns error if they are the same.

SR.NO.	METHODS & DESCRIPTION
1	**void assertEquals(boolean expected, boolean actual)** Checks that two primitives/objects are equal.
2	**void assertTrue(boolean expected, boolean actual)** Checks that a condition is true.
3	**void assertFalse(boolean condition)** Checks that a condition is false.
4	**void assertNotNull(Object object)** Checks that an object isn't null.
5	**void assertNull(Object object)** Checks that an object is null.
6	**void assertSame(boolean condition)** The assertSame() method tests if two object references point to the same object.
7	**void assertNotSame(boolean condition)** The assertNotSame() method tests if two object references do not point to the same object.
8	**void assertArrayEquals(expectedArray, resultArray);** The assertArrayEquals() method will test whether two arrays are equal to each other.

Write the “equals” method in Money.class

- The equals method in Object returns true when both objects are the same. However, Money is a *value object*. Two Monies are considered equal if they have the same currency and value.

Write the "equals" method in Money.class

```
public boolean equals(Object anObject) {
    if (anObject instanceof Money) {
        Money aMoney= (Money)anObject;
        return aMoney.currency().equals(currency())
            && amount() == aMoney.amount();
    }
    return false;
}
```

With an equals method in hand we can verify the outcome of testSimpleAdd. In JUnit you do so by a calling **Assert.assertTrue**, which triggers a failure that is recorded by JUnit when the argument isn't true.

SETUP METHOD FOR TESTING

- Now that we have implemented two test cases we notice some code duplication for setting-up the tests. It would be nice to reuse some of this test set-up code. In other words, we would like to have a common fixture for running the tests. With JUnit you can do so by storing the fixture's objects in instance variables of your **TestCase** subclass and initialize them by overridding the setUp method. The symmetric operation to setUp is tearDown which you can override to clean up the test fixture at the end of a test. Each test runs in its own fixture and JUnit calls setUp and tearDown for each test so that there can be no side effects among test runs.

MoneyTest.class

```
public class MoneyTest extends TestCase {
    private Money f12CHF;
    private Money f14CHF;

    protected void setUp() {
        f12CHF= new Money(12, "CHF");
        f14CHF= new Money(14, "CHF");
    }

    public void testEquals() {
        Assert.assertTrue(!f12CHF.equals(null));
        Assert.assertEquals(f12CHF, f12CHF);
        Assert.assertEquals(f12CHF, new Money(12, "CHF"));
        Assert.assertTrue(!f12CHF.equals(f14CHF));
    }

    public void testSimpleAdd() {
        Money expected= new Money(26, "CHF");
        Money result= f12CHF.add(f14CHF);
        Assert.assertTrue(expected.equals(result));
    }
}
```

setUp method for initializing inputs

RUNNING TESTS STATICALLY OR DYNAMICALLY

Two additional steps are needed to run the two test cases:

- define how to run an individual test case,
- define how to run a *test suite*.

JUnit supports two ways of running single tests:

- static
- dynamic

Calling tests statically

- In the static way you override the runTest method inherited from TestCase and call the desired test case. A convenient way to do this is with an anonymous inner class. Note that each test must be given a name, so you can identify it if it fails.

```
TestCase test= new MoneyTest("simple add") {
    public void runTest() {
        testSimpleAdd();
    }
}
```

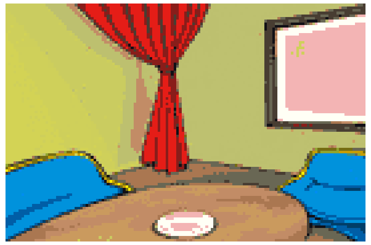
Calling Tests Dynamically

The dynamic way to create a test case to be run uses reflection to implement runTest. It assumes the name of the test is the name of the test case method to invoke. It dynamically finds and invokes the test method. To invoke the testSimpleAdd test we therefore construct a MoneyTest as shown below:

```
TestCase test= new MoneyTest("testSimpleAdd");
```

The dynamic way is more compact to write but it is less static type safe. An error in the name of the test case goes unnoticed until you run it and get a NoSuchMethodException. Since both approaches have advantages, we decided to leave the choice of which to use up to you.

TEST SUITES IN JUNIT



- As the last step to getting both test cases to run together, we have to define a test suite. In JUnit this requires the definition of a static method called suite. The suite method is like a main method that is specialized to run tests. Inside suite you add the tests to be run to a **TestSuite** object and return it. A TestSuite can run a collection of tests. TestSuite and TestCase both implement an interface called Test which defines the methods to run a test. This enables the creation of test suites by composing arbitrary TestCases and TestSuites. In short TestSuite is a Composite [1]. The next code illustrates the creation of a test suite with the dynamic way to run a test.

TEST SUITES IN JUNIT

```
public static Test suite() {
    TestSuite suite= new TestSuite();
    suite.addTest(new MoneyTest("testEquals"));
    suite.addTest(new MoneyTest("testSimpleAdd"));
    return suite;
}
```

Since JUnit 2.0 there is an even simpler dynamic way. You only pass the class with the tests to a TestSuite and it extracts the test methods automatically.

```
public static Test suite() {
   return new TestSuite(MoneyTest.class);
}
```

Static Test Suite

Here is the corresponding code using the static way.

```
public static Test suite() {
    TestSuite suite= new TestSuite();
    suite.addTest(
        new MoneyTest("money equals") {
            protected void runTest() { testEquals(); }
        }
    );

    suite.addTest(
        new MoneyTest("simple add") {
            protected void runTest() { testSimpleAdd(); }
        }
    );
    return suite;
}
```

Running Your Tests

To run JUnit Type:

for the batch TestRunner type:

```
java junit.textui.TestRunner NameOfTest
```

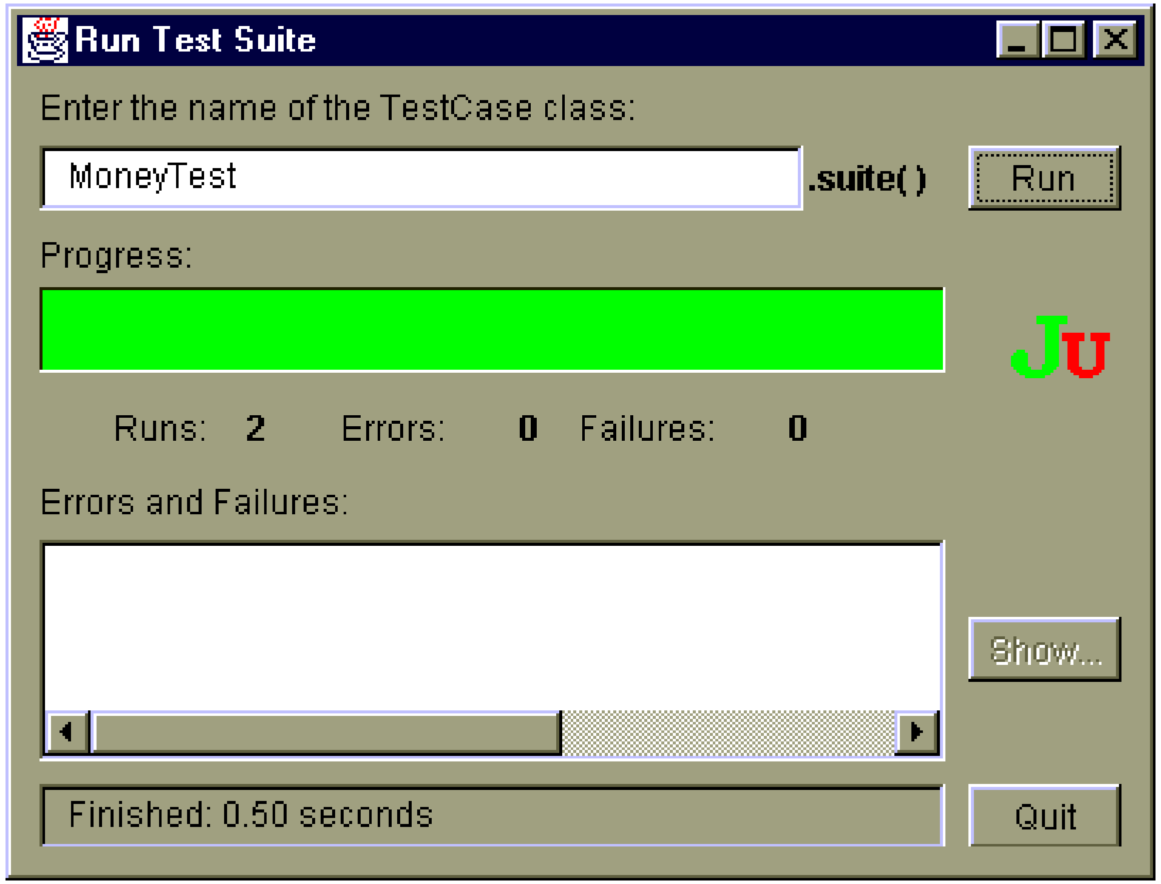
for the graphical TestRunner type:

```
java junit.awtui.TestRunner NameOfTest
```

for the Swing based graphical TestRunner type:

```
java junit.swingui.TestRunner NameOfTest
```

SUCCESSFUL TEST



Enter the name of the TestCase class:

MoneyTest .suite( )

Run

Progress:

Runs: 2 Errors: 0 Failures: 0

Errors and Failures:

Finished: 0.50 seconds

When the test results are valid and no errors are found, the progress bar will be completely green. If there are 1 or more errors, the progress bar will turn red. The errors and failures box will notify you to where to bug has occurred.

Example continued (Money Bags)

- After having verified that the simple currency case works we move on to multiple currencies. As mentioned above the problem of mixed currency arithmetic is that there isn't a single exchange rate. To avoid this problem we introduce a *MoneyBag* which defers exchange rate conversions.

Example:

- Adding 12 Swiss Francs to 14 US Dollars is represented as a bag containing the two Monies 12 CHF and 14 USD.
- Adding another 10 Swiss francs gives a bag with 22 CHF and 14 USD.

MoneyBag.class

```
class MoneyBag {
    private Vector fMonies= new Vector();

    MoneyBag(Money m1, Money m2) {
        appendMoney(m1);
        appendMoney(m2);
    }

    MoneyBag(Money bag[]) {
        for (int i= 0; i < bag.length; i++)
            appendMoney(bag[i]);
    }
}
```

appendMoney is an internal helper method that adds a Money to the list of Moneys and takes care of consolidating Monies with the same currency

MoneyBag Test

- We skip the implementation of equals and only show the testBagEquals method. In a first step we extend the fixture to include two MoneyBags.

```
protected void setUp() {
    f12CHF= new Money(12, "CHF");
    f14CHF= new Money(14, "CHF");
    f7USD=  new Money( 7, "USD");
    f21USD= new Money(21, "USD");
    fMB1= new MoneyBag(f12CHF, f7USD);
    fMB2= new MoneyBag(f14CHF, f21USD);
}
//With this fixture the testBagEquals test case becomes:
public void testBagEquals() {
    Assert.assertTrue(!fMB1.equals(null));
    Assert.assertEquals(fMB1, fMB1);
    Assert.assertTrue(!fMB1.equals(f12CHF));
    Assert.assertTrue(!f12CHF.equals(fMB1));
    Assert.assertTrue(!fMB1.equals(fMB2));
}
```

Fixing the add method

- Following "code a little, test a little" we run our extended test with JUnit and verify that we are still doing fine. With MoneyBag in hand, we can now fix the add method in Money.

```
public Money add(Money m) {
    if (m.currency().equals(currency()) )
        return new Money(amount()+m.amount(),
currency());
    return new MoneyBag(this, m);
}
```

- As defined above this method will not compile since it expects a Money and not a MoneyBag as its return value.

IMONEY INTERFACE

- With the introduction of MoneyBag there are now two representations for Moneys which we would like to hide from the client code. To do so we introduce an interface IMoney that both representations implement. Here is the IMoney interface:

```
interface IMoney {
    public abstract IMoney add(IMoney aMoney);
    //…
}
```

More Testing

- To fully hide the different representations from the client we have to support arithmetic between all combinations of Moneys with MoneyBags. Before we code on, we therefore define a couple more test cases. The expected MoneyBag results use the convenience constructor shown above, initializing a MoneyBag from an array.

```
public void testMixedSimpleAdd() {
    // [12 CHF] + [7 USD] == {[12 CHF][7 USD]}
    Money bag[]= { f12CHF, f7USD };
    MoneyBag expected= new MoneyBag(bag);
    Assert.assertEquals(expected, f12CHF.add(f7USD));
}
```

Update the Test Suite

The other tests follow the same pattern:

- testBagSimpleAdd - to add a MoneyBag to a simple Money
- testSimpleBagAdd - to add a simple Money to a MoneyBag
- testBagBagAdd - to add two MoneyBags

Next, we extend our test suite accordingly:

```
public static Test suite() {
    TestSuite suite= new TestSuite();
    suite.addTest(new MoneyTest("testMoneyEquals"));
    suite.addTest(new MoneyTest("testBagEquals"));
    suite.addTest(new MoneyTest("testSimpleAdd"));
    suite.addTest(new MoneyTest("testMixedSimpleAdd"));
    suite.addTest(new MoneyTest("testBagSimpleAdd"));
    suite.addTest(new MoneyTest("testSimpleBagAdd"));
    suite.addTest(new MoneyTest("testBagBagAdd"));
    return suite;
}
```

IMPLEMENTATION

- Having defined the test cases we can start to implement them. The implementation challenge here is dealing with all the different combinations of Money with MoneyBag. Double dispatch is an elegant way to solve this problem. The idea behind double dispatch is to use an additional call to discover the kind of argument we are dealing with. We call a method on the argument with the name of the original method followed by the class name of the receiver.

The add method in Money and MoneyBag becomes:

Implementation

```
class Money implements IMoney {
    public IMoney add(IMoney m) {
        return m.addMoney(this);
    }
    //…
}
class MoneyBag implements IMoney {
    public IMoney add(IMoney m) {
        return m.addMoneyBag(this);
    }
    //…
}
```

In order to get this to compile we need to extend the interface of IMoney with the two helper methods:

```
interface IMoney {
//…
    IMoney addMoney(Money aMoney);
    IMoney addMoneyBag(MoneyBag aMoneyBag);
}
```

Implementation

- To complete the implementation of double dispatch, we have to implement these methods in Money and MoneyBag. This is the implementation in Money.

```
public IMoney addMoney(Money m) {
    if (m.currency().equals(currency()) )
        return new Money(amount()+m.amount(), currency());
    return new MoneyBag(this, m);
}

public IMoney addMoneyBag(MoneyBag s) {
    return s.addMoney(this);
}
```

IMPLEMENTATION

- Here is the implemenation in MoneyBag which assumes additional constructors to create a MoneyBag from a Money and a MoneyBag and from two MoneyBags.

```
public IMoney addMoney(Money m) {
    return new MoneyBag(m, this);
}

public IMoney addMoneyBag(MoneyBag s) {
    return new MoneyBag(s, this);
}
```

Are there more errors that can occur?

- We run the tests, and they pass. However, while reflecting on the implementation we discover another interesting case. What happens when as the result of an addition a MoneyBag turns into a bag with only one Money? For example, adding -12 CHF to a Moneybag holding 7 USD and 12 CHF results in a bag with just 7 USD. Obviously, such a bag should be equal with a single Money of 7 USD. To verify the problem you can make a test and run it.

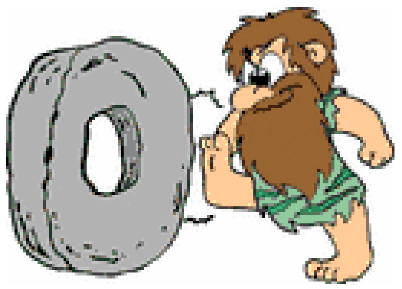
REVIEW

- We wrote the first test, testSimpleAdd, immediately after we had written add(). In general, your development will go much smoother if you write tests a little at a time as you develop. It is at the moment that you are coding that you are imagining how that code will work. That's the perfect time to capture your thoughts in a test.

REVIEW

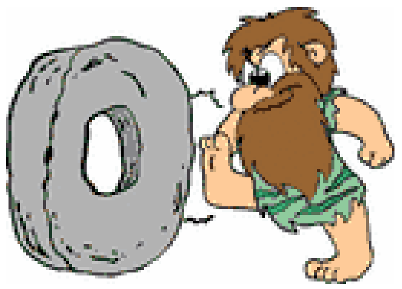
- We refactored the existing tests, testSimpleAdd and testEqual, as soon as we introduced the common setUp code. Test code is just like model code in working best if it is factored well. When you see you have the same test code in two places, try to find a way to refactor it so it only appears once.
- We created a suite method, and then extended it when we applied Double Dispatch. Keeping old tests running is just as important as making new ones run. The ideal is to always run all of your tests. Sometimes that will be too slow to do 10 times an hour. Make sure you run all of your tests at least daily.

TESTING PRACTICES



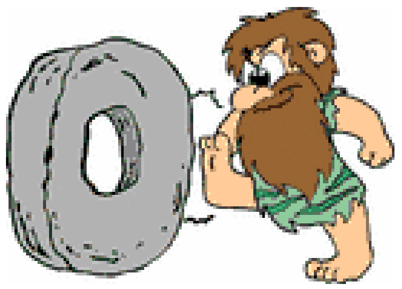
- Martin Fowler makes this easy for you. He says, "Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." At first you will find that you have to create new fixtures all the time, and testing will seem to slow you down a little. Soon, however, you will begin reusing your library of fixtures and new tests will usually be as simple as adding a method to an existing TestCase subclass

Testing Practices



- You can always write more tests. However, you will quickly find that only a fraction of the tests you can imagine are actually useful. What you want is to write tests that fail even though you think they should work, or tests that succeed even though you think they should fail. Another way to think of it is in cost/benefit terms. You want to write tests that will pay you back with information.

Testing Practices

Here are a couple of the times that you will receiv  eturn on your testing investment:

- During Development- When you need to add new functionality to the system, write the tests first. Then, you will be done developing when the test runs.
- During Debugging- When someone discovers a defect in your code, first write a test that will succeed if the code is working. Then debug until the test succeeds.

Maven errors occurred:

- 1) During running test: Nothing to test.

---Solution: write at least some code/method in the class.

2) Test failure: //fail("Not yet implemented"); commenting out this line

3) Runtime error. Can not find jre.

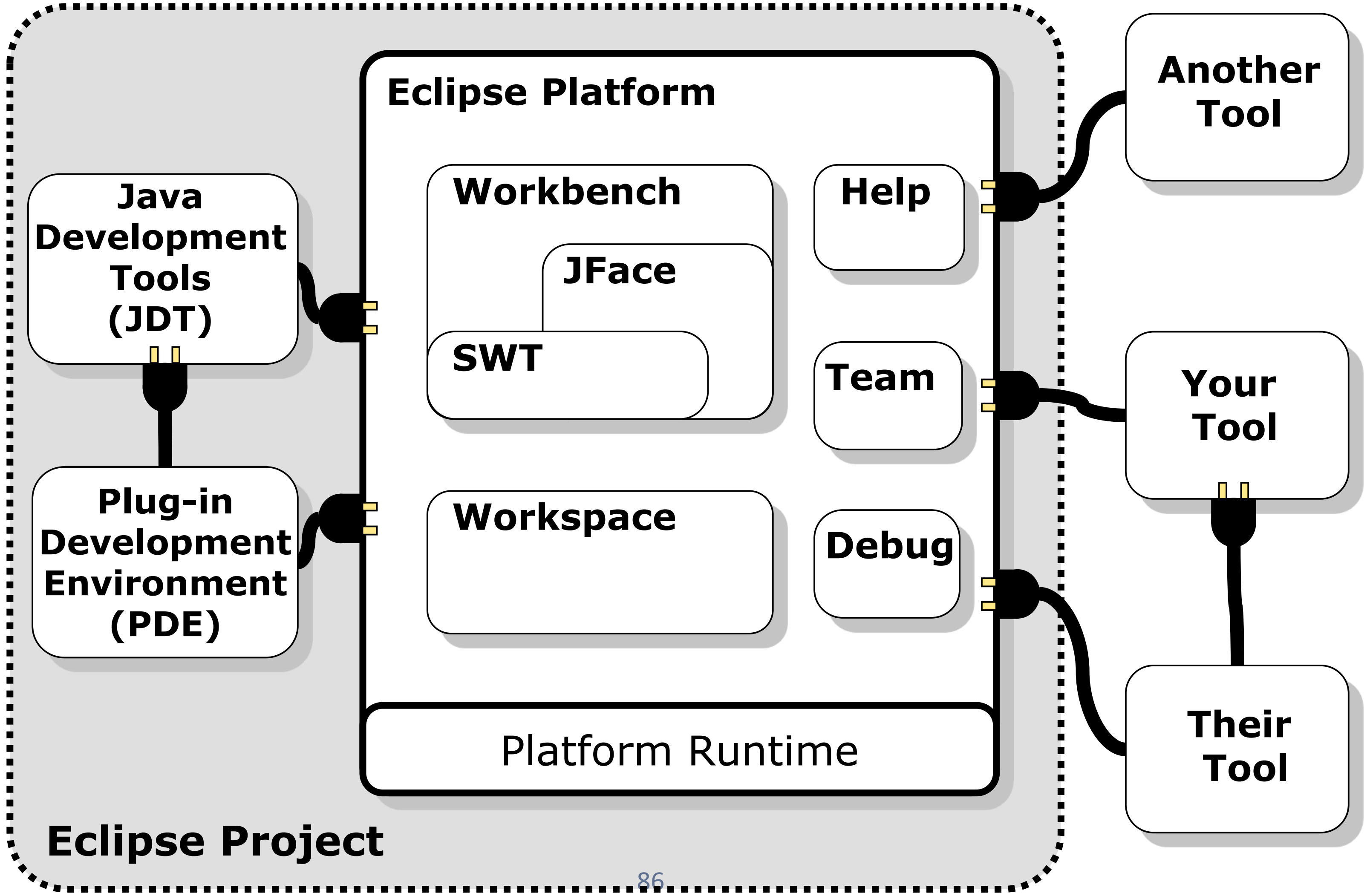
---Solution: the jre file might be corrupted. Try deleting that and download new.

4) Eclipse General Skills and Practice

ECLIPSE PROJECT AIMS

- Provide open platform for application development tools
  - Run on a wide range of operating systems
  - GUI and non-GUI
- Language-neutral
  - Permit unrestricted content types
  - HTML, Java, C, JSP, EJB, XML, GIF, …
- Facilitate seamless tool integration
  - At UI and deeper
  - Add new tools to existing installed products
- Attract community of tool developers
  - Including independent software vendors (ISVs)
  - Capitalize on popularity of Java for writing tools

ECLIPSE OVERVIEW

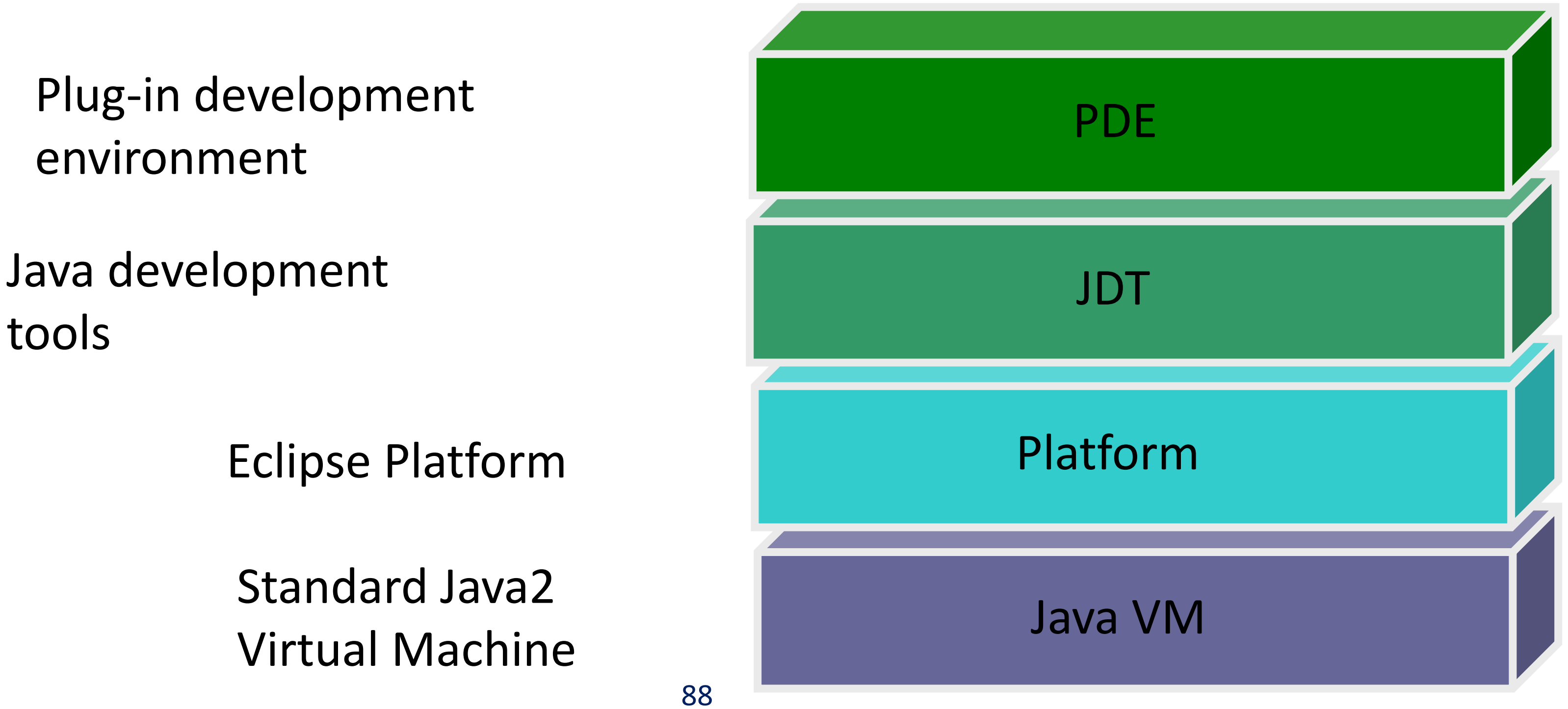


Eclipse Origins

- Eclipse created by OTI and IBM teams responsible for IDE products
  - IBM VisualAge/Smalltalk (Smalltalk IDE)
  - IBM VisualAge/Java (Java IDE)
  - IBM VisualAge/Micro Edition (Java IDE)
- Initially staffed with 40 full-time developers
- Geographically dispersed development teams
  - OTI Ottawa, OTI Minneapolis, OTI Zurich, IBM Toronto, OTI Raleigh, IBM RTP, IBM St. Nazaire (France)
- Effort transitioned into open source project
  - IBM donated initial Eclipse code base
    - Platform, JDT, PDE

What is Eclipse?

- Eclipse is a universal platform for integrating development tools
- Open, extensible architecture based on plug-ins


Plug-in development environment
PDE
Java development tools
JDT
Eclipse Platform
Platform
Standard Java2 Virtual Machine
Java VM

Eclipse Plug-in Architecture

- **Plug-in -** smallest unit of Eclipse function
  - Big example: HTML editor
  - Small example: Action to create zip files
- **Extension point -** named entity for collecting “contributions”
  - Example: extension point for workbench preference UI
- **Extension -** a contribution
  - Example: specific HTML editor preferences

ECLIPSE PLUG-IN ARCHITECTURE

- Each plug-in
  - Contributes to 1 or more extension points
  - Optionally declares new extension points
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export Java-based APIs for downstream plug-ins
  - Lives in its own plug-in subdirectory

- Details spelled out in the **plug-in manifest**
  - Manifest declares contributions
  - Code implements contributions and provides API
  - plugin.xml file in root of plug-in subdirectory

Plug-in Manifest

plugin.xml

```
    id = “com.example.tool"
    name = “Example Plug-in Tool"
    class = "com.example.tool.ToolPlugin">
  <requires>
    <import plugin = "org.eclipse.core.resources"/>
    <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
    <library name = “tool.jar"/>
  </runtime>
  <extension
    point = "org.eclipse.ui.preferencepages">
   <page id = "com.example.tool.preferences"
     icon = "icons/knob.gif"
     title = “Tool Knobs"
     class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
    name = “Frob Providers“
    id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

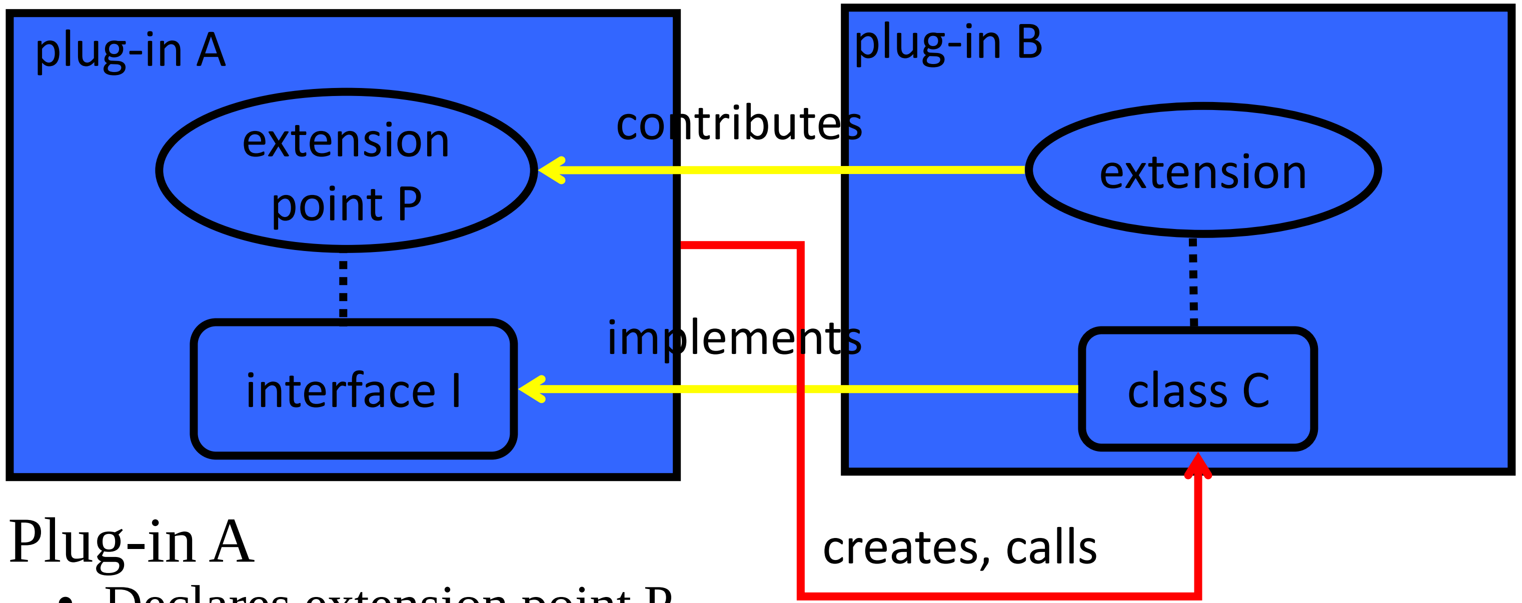
Location of plug-in’s code

Declare contribution this plug-in makes

Declare new extension point open to contributions from other plug-ins

Eclipse Plug-in Architecture

- Typical arrangement



- Plug-in A
  - Declares extension point P
  - Declares interface I to go with P
- Plug-in B
  - Implements interface I with its own class C
  - Contributes class C to extension point P
- Plug-in A instantiates C and calls its I methods

Eclipse Platform Architecture

- Eclipse Platform Runtime is micro-kernel
  - All functionality supplied by plug-ins

- Eclipse Platform Runtime handles start up
  - Discovers plug-ins installed on disk
  - Matches up extensions with extension points
  - Builds global plug-in registry
  - Caches registry on disk for next time

Plug-in Activation

- Each plug-in gets its own Java class loader
  - Delegates to required plug-ins
  - Restricts class visibility to exported APIs

- Contributions processed without plug-in activation
  - Example: Menu constructed from manifest info for contributed items

- Plug-ins are activated only as needed
  - Example: Plug-in activated only when user selects its menu item
  - Scalable for large base of installed plug-ins
  - Helps avoid long start up times

Plug-in Fragments

- **Plug-in fragments** holds some of plug-in’s files
  - Separately installable
- Each fragment has separate subdirectory
  - Separate manifest file
- Logical plug-in = Base plug-in + fragments
- Plug-in fragments used for
  - Isolation of OS dependencies
  - Internalization – fragments hold translations

Plug-in Install

- **Features** group plug-ins into installable chunks
  - Feature manifest file
- Plug-ins and features bear version identifiers
  - major . minor . service
  - Multiple versions may co-exist on disk
- Features downloadable from web site
  - Using Eclipse Platform update manager
  - Obtain and install new plug-ins
  - Obtain and install updates to existing plug-ins

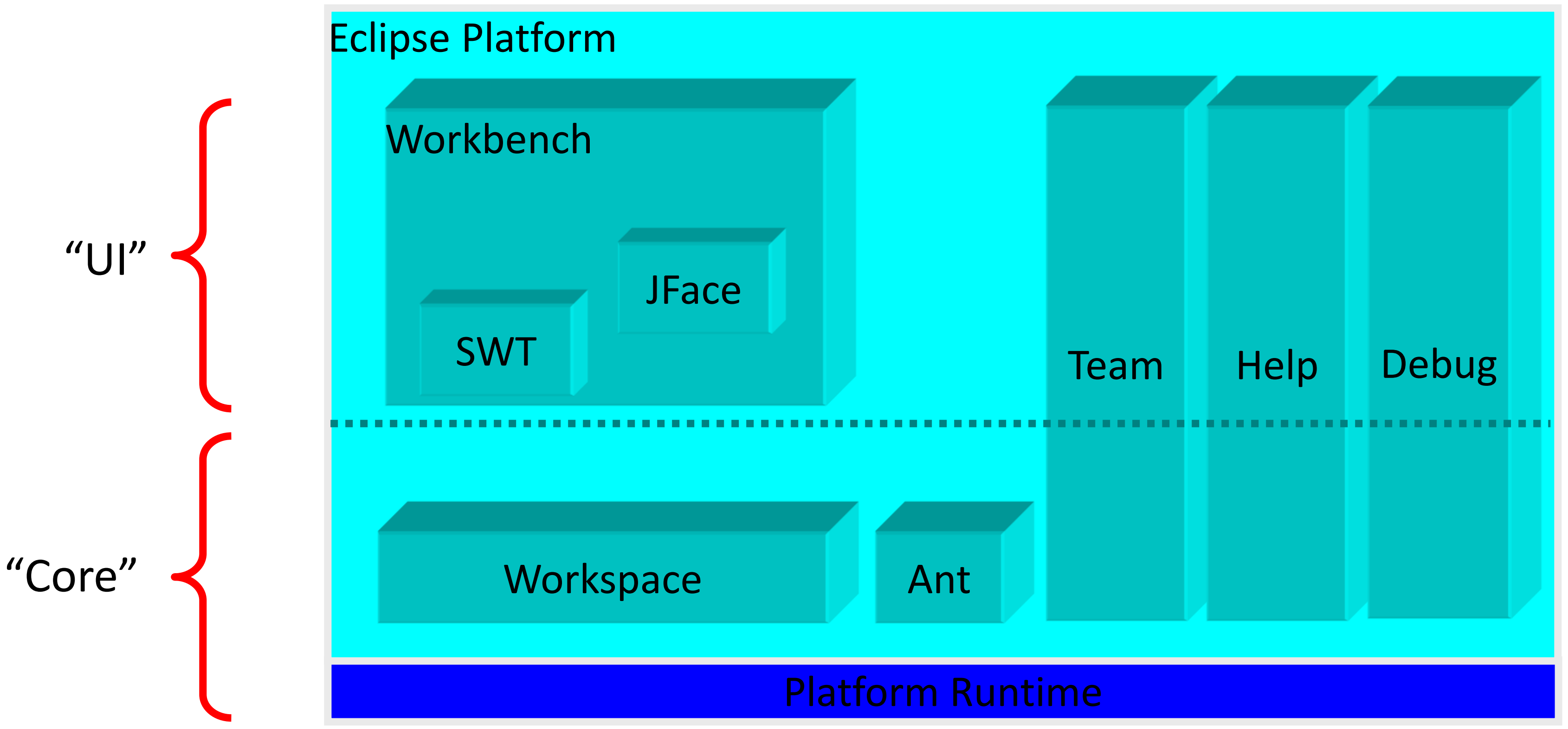
Plug-in Architecture - Summary

- All functionality provided by plug-ins
  - Includes all aspects of Eclipse Platform itself
- Communication via extension points
  - Contributing does not require plug-in activation
- Packaged into separately installable features
  - Downloadable

Eclipse has open, extensible architecture based on plug-ins

ECLIPSE PLATFORM

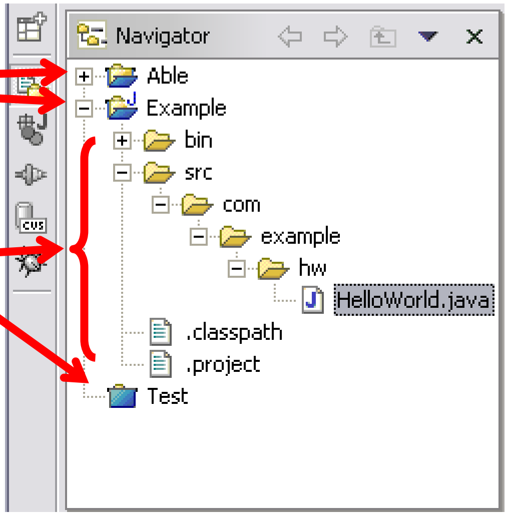
- Eclipse Platform is the common base
- Consists of several key components



Workspace Component

- Tools operate on files in user's **workspace**

- Workspace holds 1 or more top-level **projects**
- Projects map to directories in file system
- Tree of **folders** and **files**
- {Files, Folders, Projects} termed **resources**



- Tools read, create, modify, and delete resources in workspace
- Plug-ins access via workspace and resource APIs

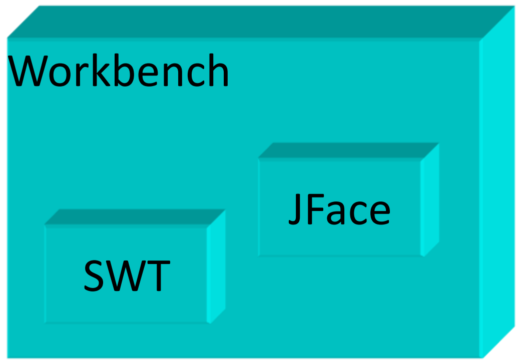
Workspace and Resource API

- Allows fast navigation of workspace resource tree
- Resource change listener for monitoring activity
  - Resource deltas describe batches of changes
- Maintains limited history of changed/deleted files
- Several kinds of extensible resource metadata
  - Persistent resource properties
  - Session resource properties
  - Markers
  - Project natures
- Workspace session lifecycle
  - Workspace save, exit, restore
- Incremental project builders

INCREMENTAL PROJECT BUILDERS

- Problem: coordinated analysis and transformation of thousands of files
  - Compiling all source code files in project
  - Checking for broken links in HTML files
- Scalable solution requires incremental reanalysis
- Incremental project builder API/framework
  - Builders are passed resource delta
  - Delta describes all changes since previous build
  - Basis for incremental tools
- Extensible – plug-ins define new types of builders
  - JDT defines Java builder
- Configurable – any number of builders per project

Workbench Component



- SWT – generic low-level graphics and widget set
- JFace – UI frameworks for common UI tasks
- Workbench – UI personality of Eclipse Platform

SWT

- SWT = Standard Widget Toolkit
- Generic graphics and GUI widget set
  - buttons, lists, text, menus, trees, styled text...

- Simple
- Small
- Fast
- OS-independent API
- Uses native widgets where available
- Emulates widgets where unavailable

WHY SWT?

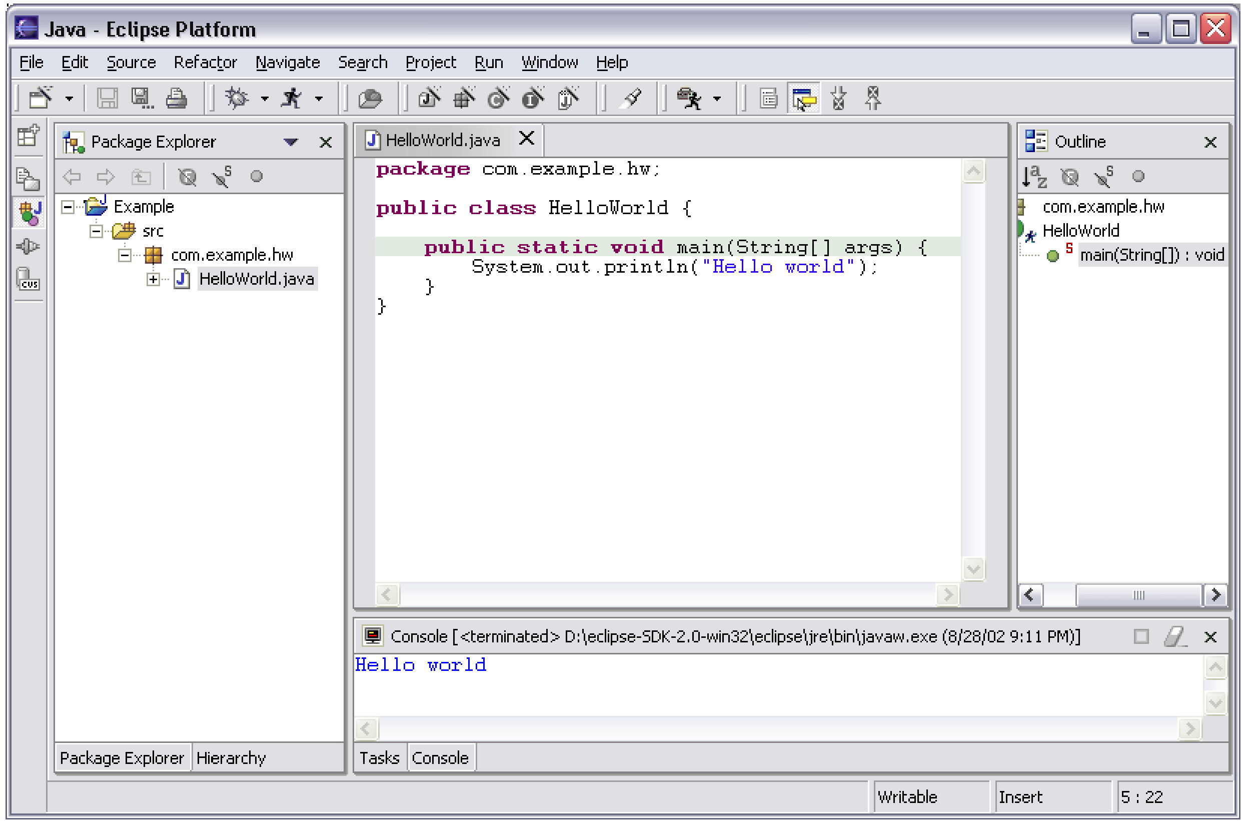
- Consensus: hard to produce professional looking shrink-wrapped products using Swing and AWT

- SWT provides
  - Tight integration with native window system
  - Authentic native look and feel
  - Good performance
  - Good portability
  - Good base for robust GUIs

- The proof of the pudding is in the eating…

WHY SWT?

- Eclipse Platform on Windows XP



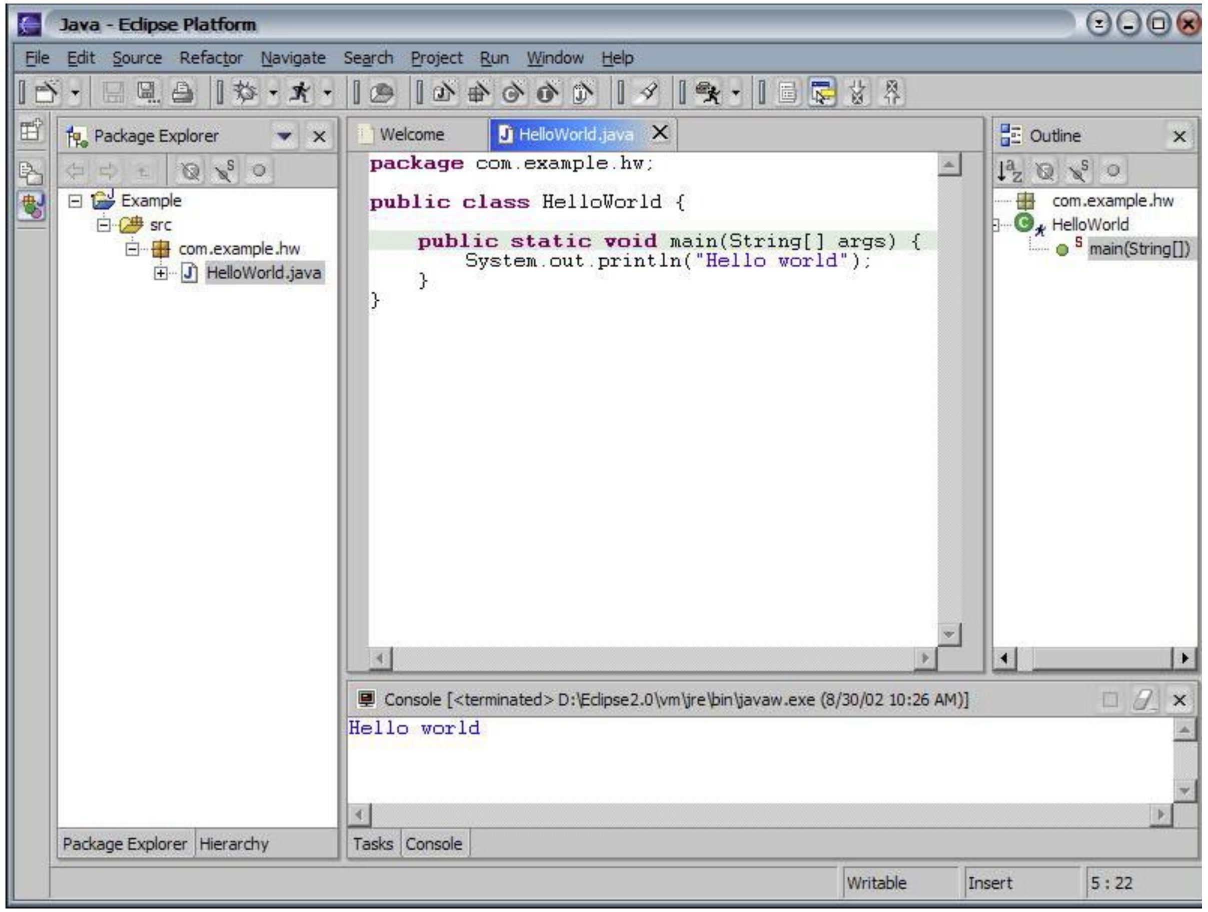
```java
package com.example.hw;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Console [<terminated> D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (8/28/02 9:11 PM)]

```
Hello world
```

WHY SWT?

- Eclipse Platform on Windows XP (skinned)



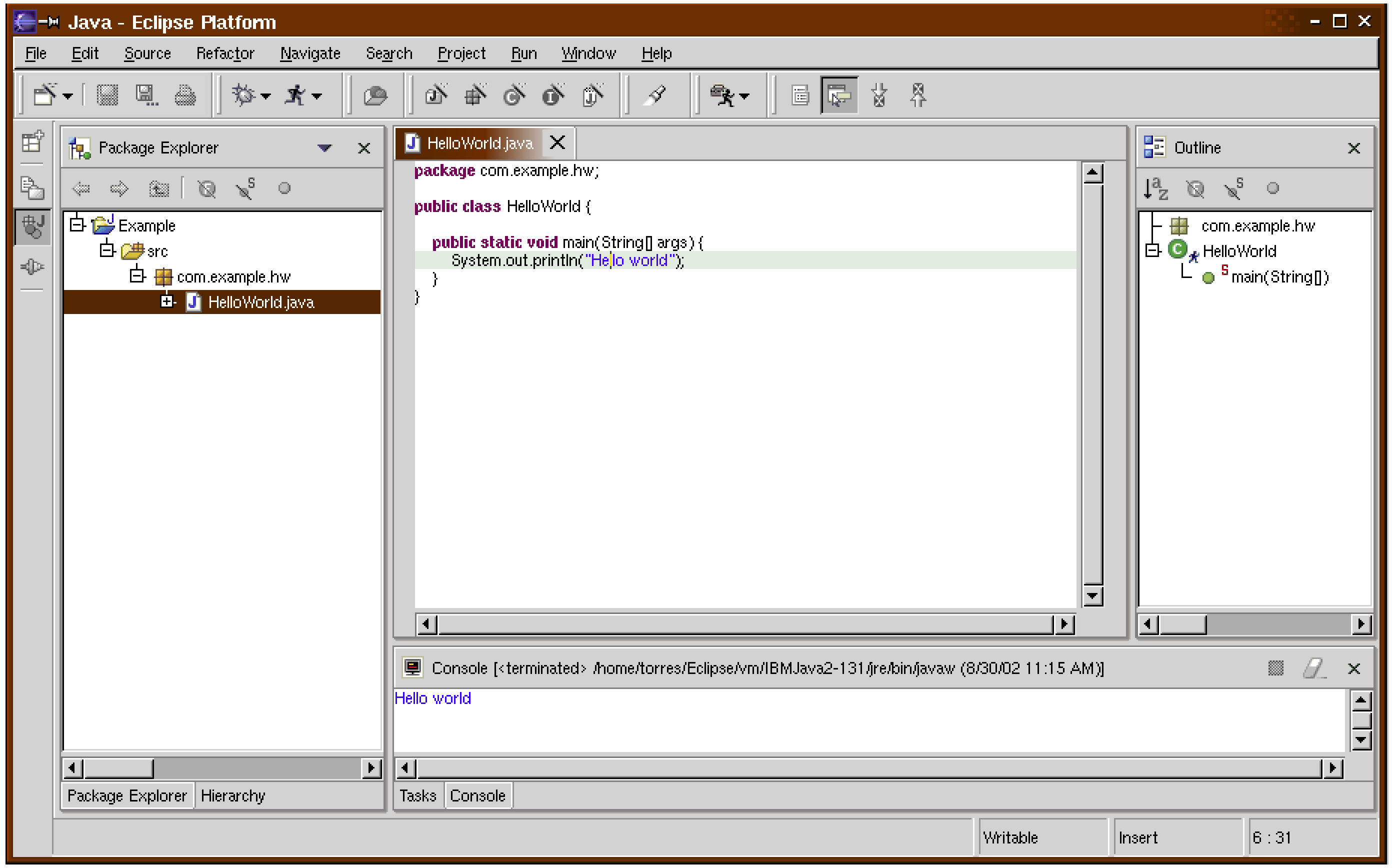
```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Console [<terminated> D:\Eclipse2.0\vm\jre\bin\javaw.exe (8/30/02 10:26 AM)]

```
Hello world
```

WHY SWT?

- Eclipse Platform on Linux - GTK 2.0



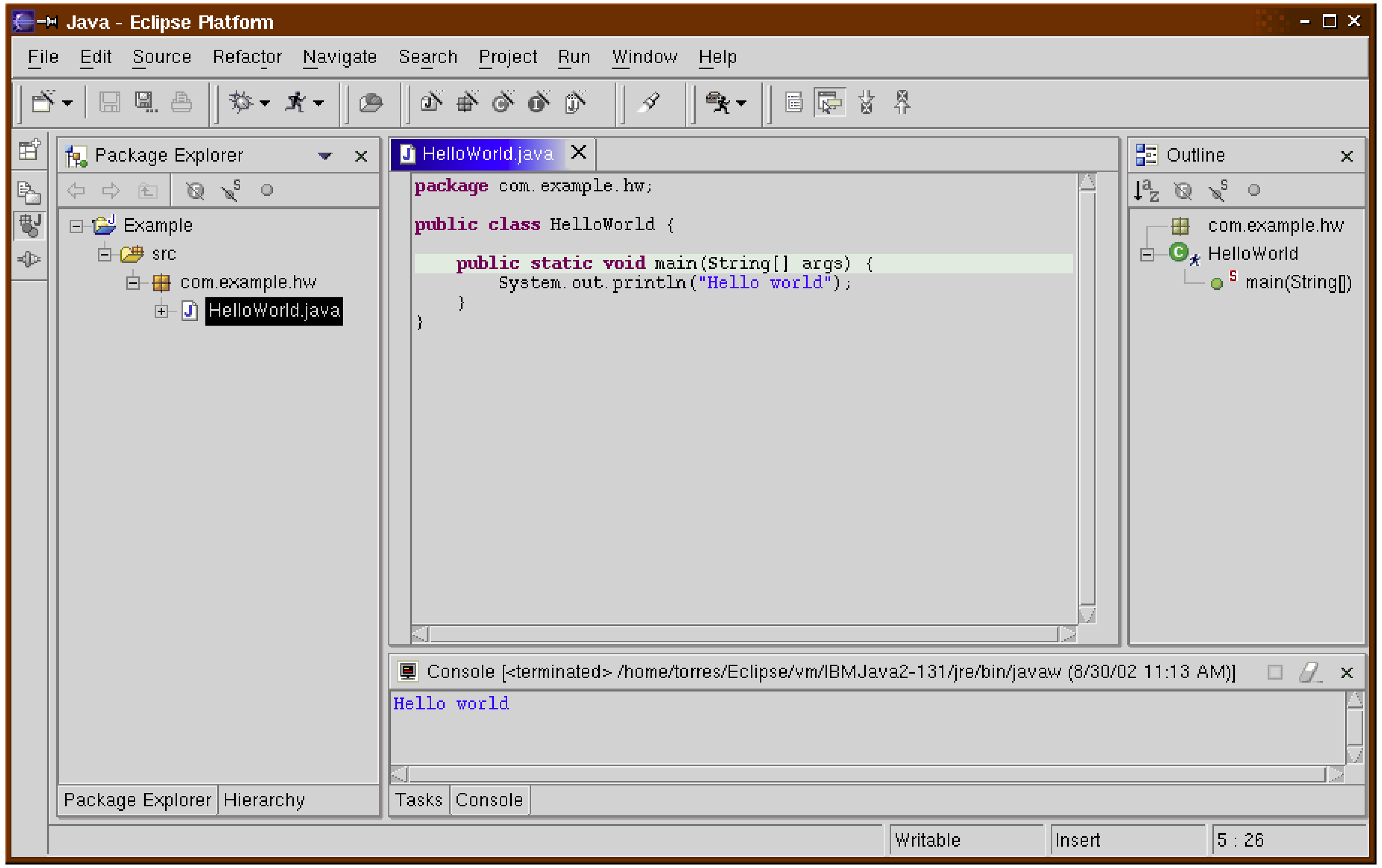
```
package com.example.hw;

public class HelloWorld {

  public static void main(String[] args) {
    System.out.println("Hello world");
  }
}
```

Console [<terminated> /home/torres/Eclipse/vm/IBMJava2-131/jre/bin/javaw (8/30/02 11:15 AM)]

```
Hello world
```

WHY SWT?

- Eclipse Platform on Linux - Motif



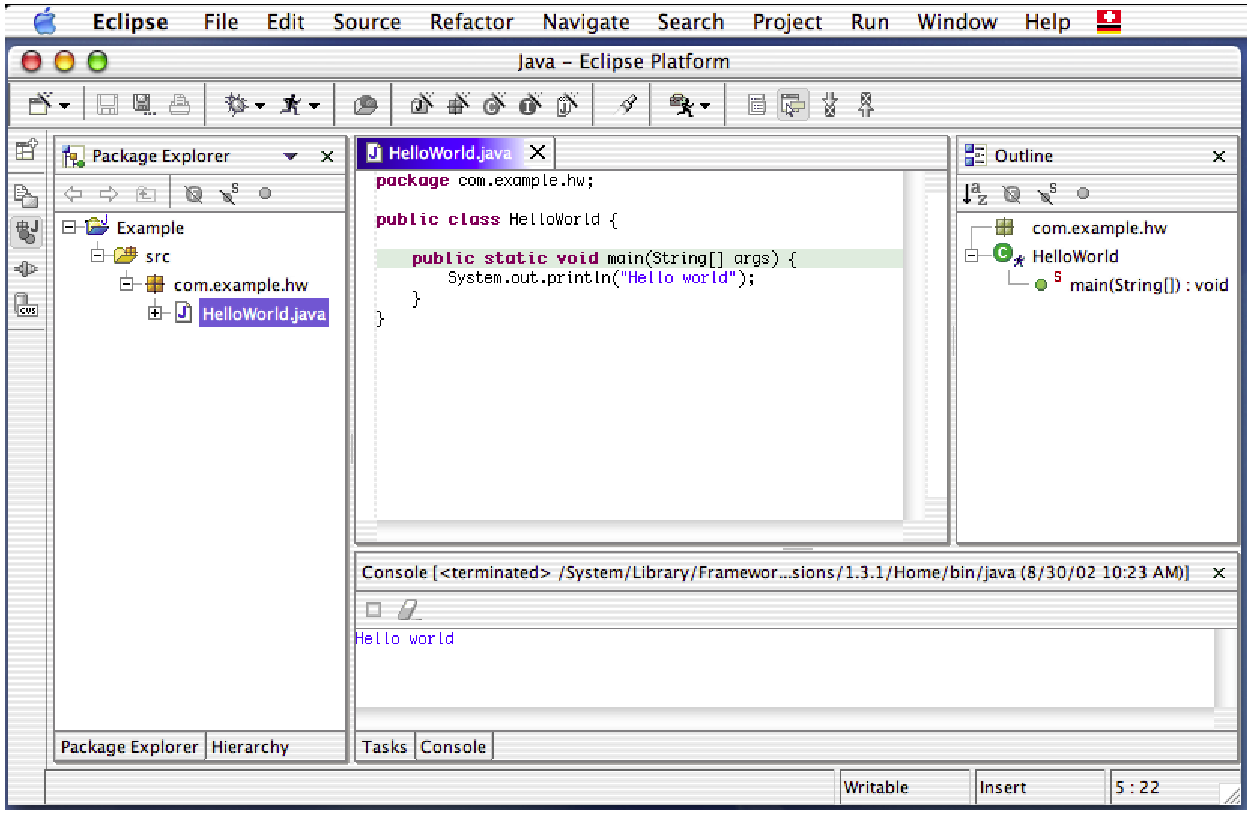
```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Console [<terminated> /home/torres/Eclipse/vm/IBMJava2-131/jre/bin/javaw (8/30/02 11:13 AM)]

```
Hello world
```

WHY SWT?

- Eclipse Platform on Mac OS X - Carbon



```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Console [<terminated> /System/Library/Framewor...sions/1.3.1/Home/bin/java (8/30/02 10:23 AM)]

```
Hello world
```

JFace

- JFace is set of UI frameworks for common UI tasks
- Designed to be used in conjunction with SWT
- Classes for handling common UI tasks
- API and implementation are window-system independent

JFace APIs

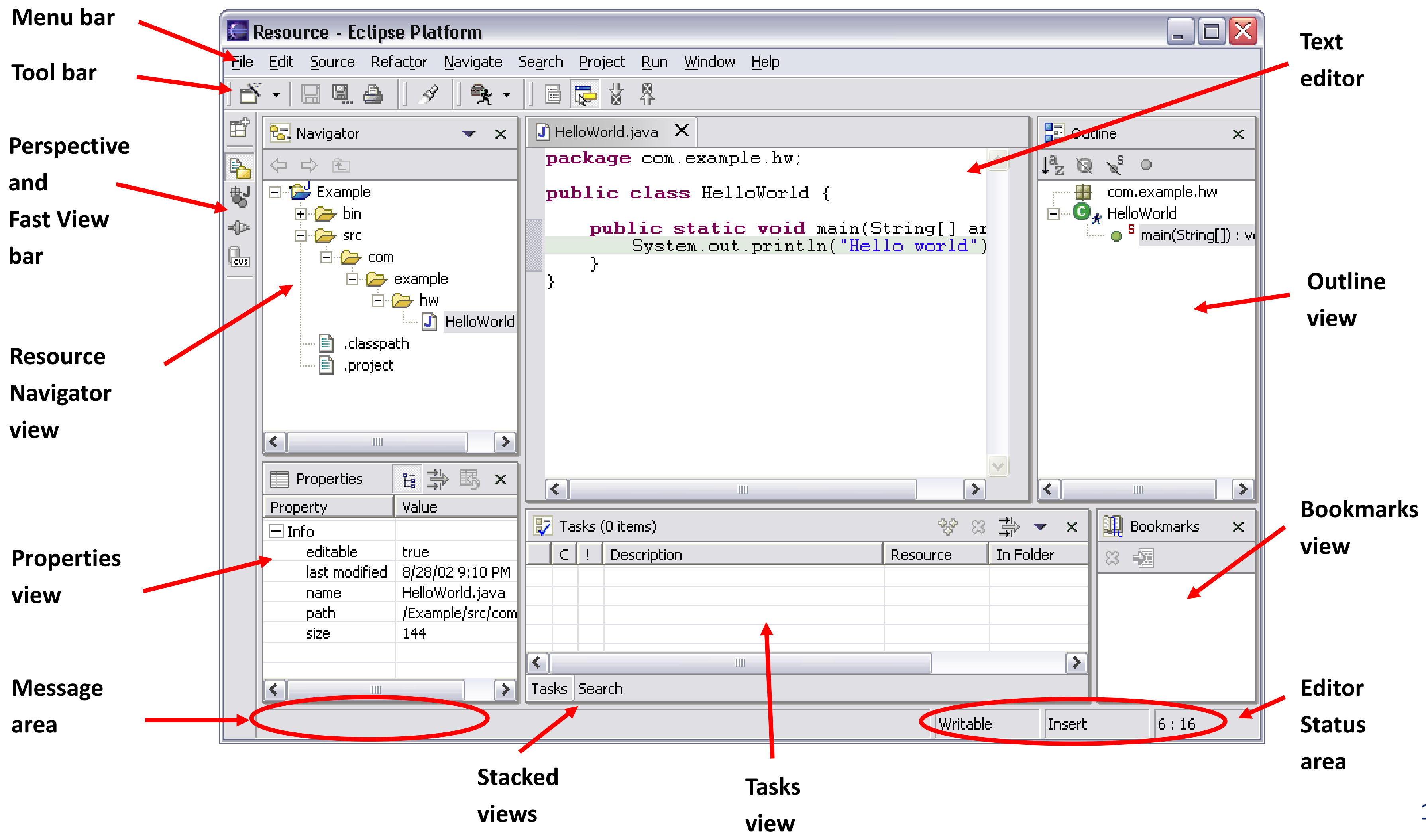
- Image and font registries
- Dialog, preference, and wizard frameworks
- Structured viewers
  - Model-aware adapters for SWT tree, table, list widgets
- Text infrastructure
  - Document model for SWT styled text widget
  - Coloring, formatting, partitioning, completion
- Actions
  - Location-independent user commands
  - Contribute action to menu, tool bar, or button

Workbench Component

- Workbench is UI personality of Eclipse Platform

- UI paradigm centered around
    - Editors
    - Views
    - Perspectives

WORKBENCH TERMINOLOGY



```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

EDITORS

- Editors appear in workbench editor area
- Contribute actions to workbench menu and tool bars
- Open, edit, save, close lifecycle
- Open editors are stacked

- Extension point for contributing new types of editors
- Example: JDT provides Java source file editor
- Eclipse Platform includes simple text file editor
- Windows only: embed any OLE document as editor
- Extensive text editor API and framework

Views

- Views provide information on some object
- Views augment editors
  - Example: Outline view summarizes content
- Views augment other views
  - Example: Properties view describes selection

- Extension point for new types of views
- Eclipse Platform includes many standard views
  - Resource Navigator, Outline, Properties, Tasks, Bookmarks, Search, …
- View API and framework
  - Views can be implemented with JFace viewers

Perspectives

- Perspectives are arrangements of views and editors
- Different perspectives suited for different user tasks
- Users can quickly switch between perspectives
- Task orientation limits visible views, actions
    - Scales to large numbers of installed tools
- Perspectives control
    - View visibility
    - View and editor layout
    - Action visibility
- Extension point for new perspectives
- Eclipse Platform includes standard perspectives
    - Resource, Debug, …
- Perspective API

Other Workbench Features

- Tools may also
  - Add global actions
  - Add actions to existing views and editors
  - Add views, action sets to existing perspectives

- Eclipse Platform is accessible
- Accessibility mechanisms available to all plug-ins

Workbench Responsibilities

- Eclipse Platform manages windows and perspectives
- Eclipse Platform creates menu and tool bars
  - Labels and icons listed in plug-in manifest
  - Contributing plug-ins not activated
- Eclipse Platform creates views and editors
  - Instantiated only as needed
- Scalable to large numbers of installed tools

Team Component

- Version and configuration management (VCM)
- Share resources with team via a **repository**
- Repository associated at project level
- Extension point for new types of repositories
- Repository provider API and framework
- Eclipse Platform includes CVS repository provider
- Available repository providers*
  - ChangeMan (Serena)
  - ClearCase (Rational)
  - CM Synergy (Telelogic)
  - PVCS (Merant)
  - Microsoft Visual Source Safe

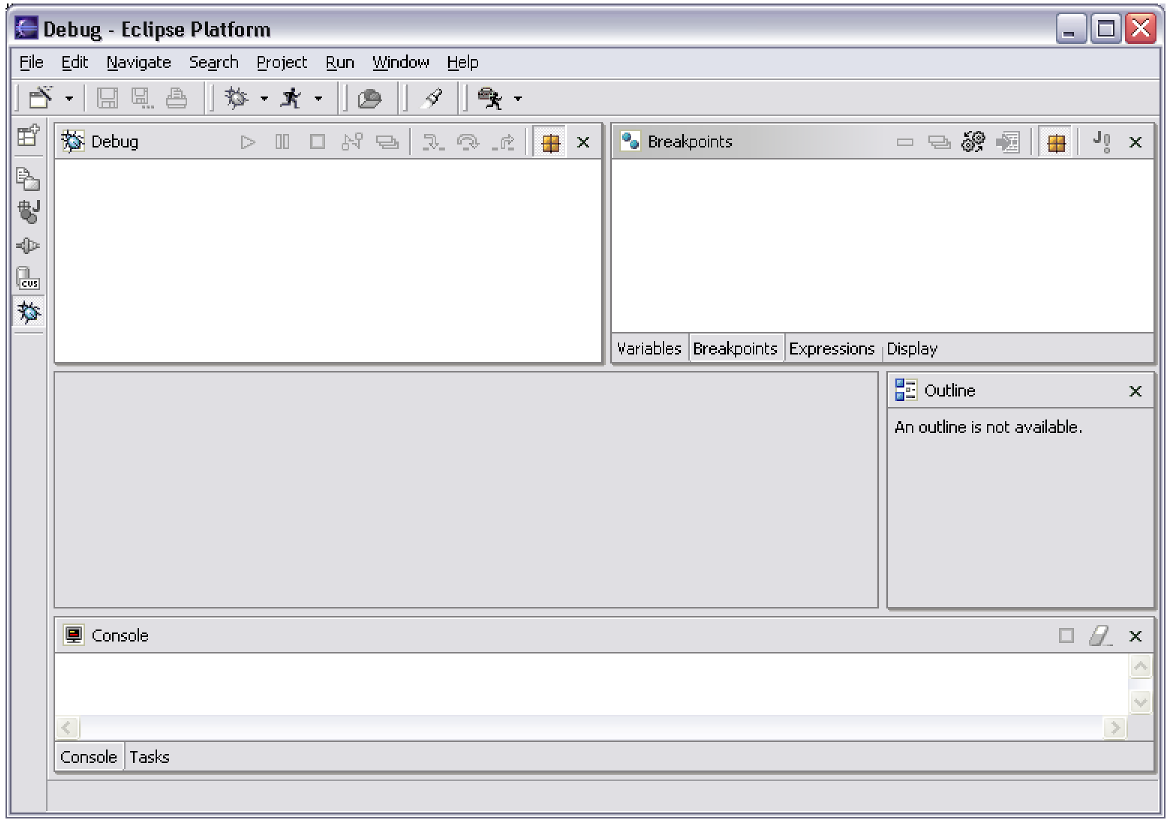
  - AllFusion Harvest (CA)
  - Perforce
  - Source Integrity (MKS)
  - TeamCode (Interwoven)

Team Component

- Repository providers have wide latitude
  - Provide actions suited to repository
  - No built-in process model
- Integrate into workbench UI via
  - Share project configuration wizard
  - Actions on Team menu
  - Resource decorators
  - Repository-specific preferences
  - Specialized views for repository browsing, …

Debug Component

- Common debug UI and underlying debug model


Debug - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
Breakpoints
Variables Breakpoints Expressions Display
Outline
An outline is not available.
Console
Console Tasks

Debug Component

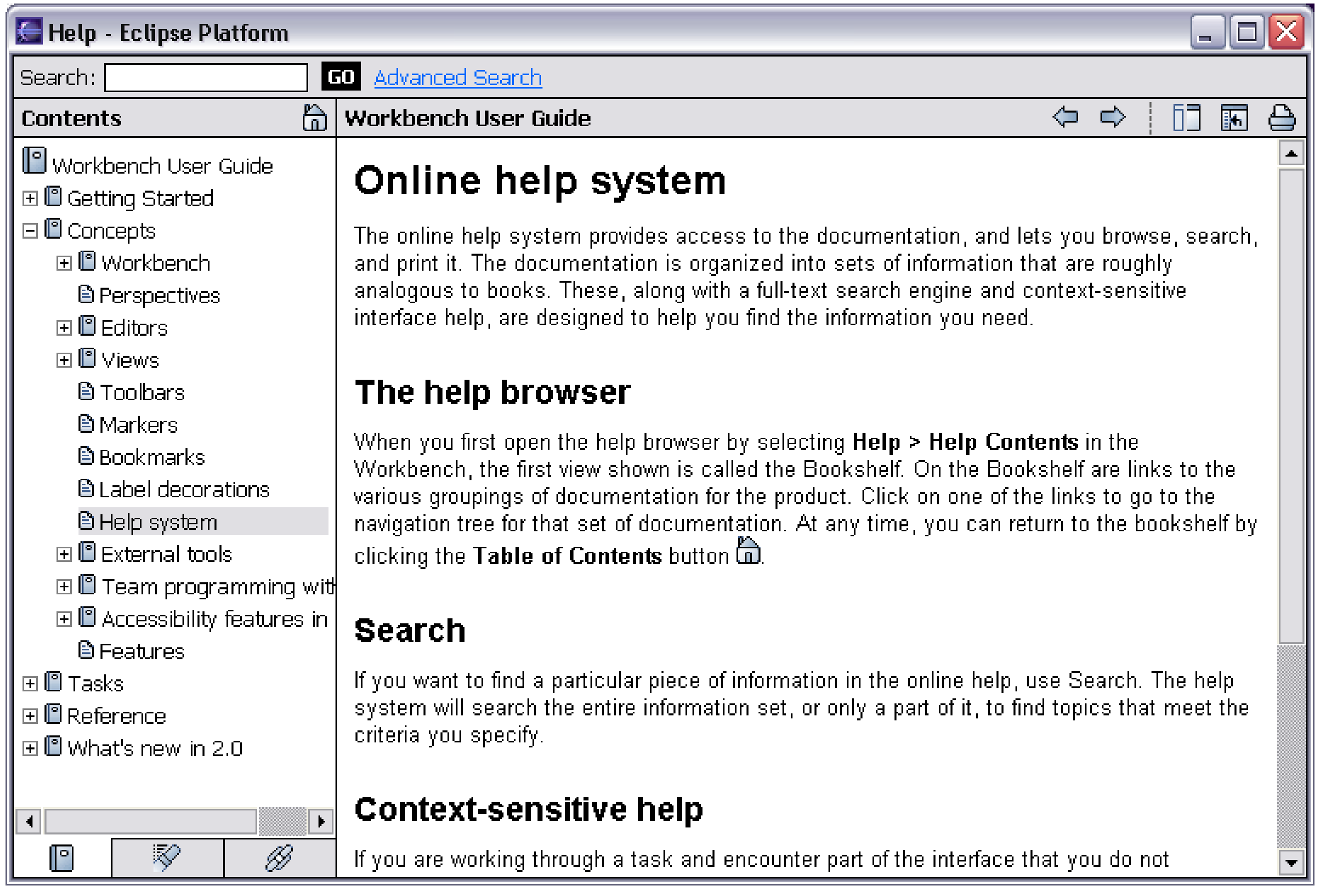
- Launch configurations
  - How to run a program (debug mode option)
- Generic debug model
  - Standard debug events: suspended, exit, …
  - Standard debug actions: resume, terminate, step, …
  - Breakpoints
  - Expressions
  - Source code locator
- Generic debug UI
  - Debug perspective
  - Debug views: stack frames, breakpoints, …
- Example: JDT supplies Java launcher and debugger
  - Java debugger based on JPDA
- Debug mechanisms available to other plug-ins

Ant Component

- Eclipse incorporates Apache Ant
- Ant is Java-based build tool
  - “Kind of like Make…without Make's wrinkles”
- XML-based build files instead of makefiles
- Available from workbench External Tools menu
- Run Ant targets in build files inside or outside workspace
- PDE uses Ant for building deployed form of plug-in

HELP COMPONENT

- Help is presented in a standard web browser


Help - Eclipse Platform

Search: GO Advanced Search

Contents | Workbench User Guide

Workbench User Guide
- Getting Started
- Concepts
  - Workbench
  - Perspectives
  - Editors
  - Views
  - Toolbars
  - Markers
  - Bookmarks
  - Label decorations
  - Help system
  - External tools
  - Team programming wit
  - Accessibility features in
  - Features
- Tasks
- Reference
- What's new in 2.0

Online help system

The online help system provides access to the documentation, and lets you browse, search, and print it. The documentation is organized into sets of information that are roughly analogous to books. These, along with a full-text search engine and context-sensitive interface help, are designed to help you find the information you need.

The help browser

When you first open the help browser by selecting **Help > Help Contents** in the Workbench, the first view shown is called the Bookshelf. On the Bookshelf are links to the various groupings of documentation for the product. Click on one of the links to go to the navigation tree for that set of documentation. At any time, you can return to the bookshelf by clicking the **Table of Contents** button .

Search

If you want to find a particular piece of information in the online help, use Search. The help system will search the entire information set, or only a part of it, to find topics that meet the criteria you specify.

Context-sensitive help

If you are working through a task and encounter part of the interface that you do not

Help Component

- Help books are HTML webs
- Extension points for contributing
  - entire books
  - sections to existing books
  - F1-help pop ups
- Eclipse Platform contributes
  - “Workbench User Guide”
  - “Platform Plug-in Developer Guide” (APIs)
  - F1-help for views, editors, dialogs, …
- JDT and PDE contribute their own help
- Help mechanisms available to all plug-ins

- Help search engine based on Apache Lucene
- Headless help server based on Apache Tomcat

INTERNATIONALIZATION

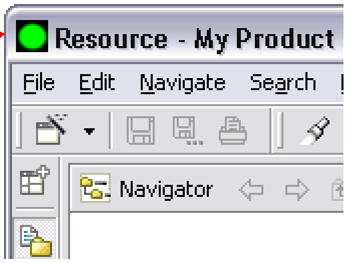
- Eclipse Platform is internationalized
- 2.0 translations available for following languages

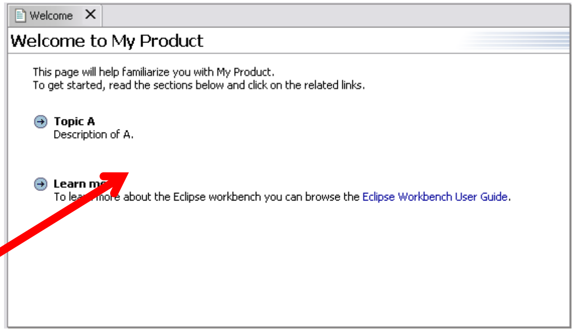
English	German
Spanish	Italian
French	Portugese (Brazil)
Japanese	Korean
Chinese (Traditional)	Chinese (Simplified)

- Translations live in plug-in fragments
  - Separately shippable
- Internalization mechanisms available to all plug-ins

PRODUCT INFORMATION

Window image




Welcome

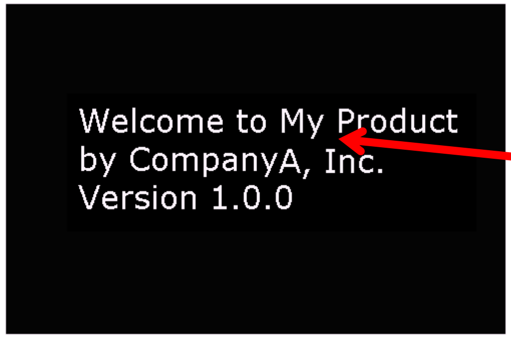
Welcome to My Product

This page will help familiarize you with My Product.
To get started, read the sections below and click on the related links.

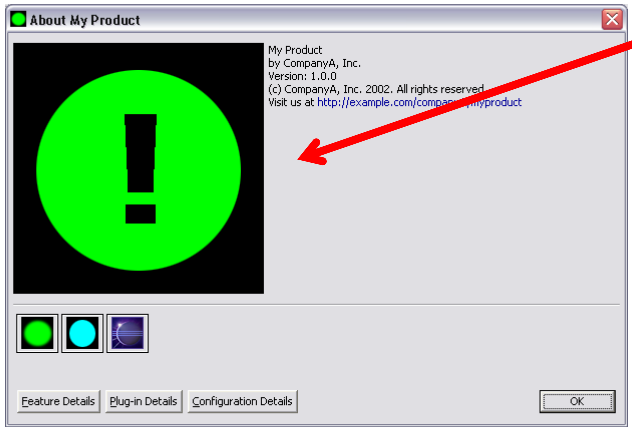
Topic A
Description of A.

Learn m
To lea more about the Eclipse workbench you can browse the Eclipse Workbench User Guide.

Welcome pages

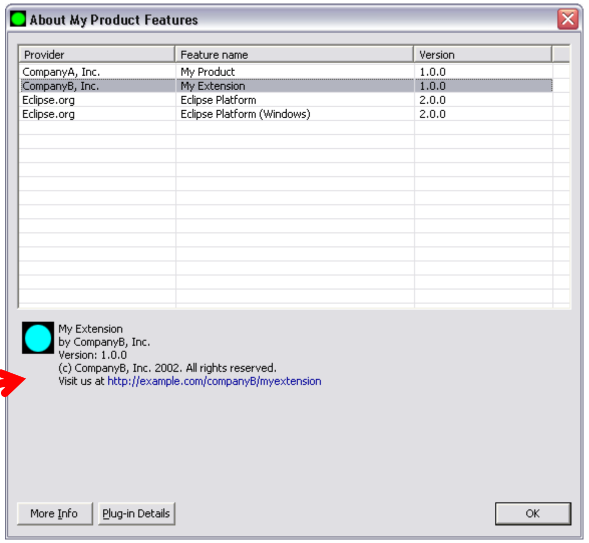

Welcome to My Product
by CompanyA, Inc.
Version 1.0.0

Splash screen


About My Product

My Product
by CompanyA, Inc.
Version: 1.0.0
(c) CompanyA, Inc. 2002. All rights reserved
Visit us at http://example.com/compan myproduct

About product info


About My Product Features

Provider	Feature name	Version
CompanyA, Inc.	My Product	1.0.0
CompanyB, Inc.	My Extension	1.0.0
Eclipse.org	Eclipse Platform	2.0.0
Eclipse.org	Eclipse Platform (Windows)	2.0.0

My Extension
by CompanyB, Inc.
Version: 1.0.0
(c) CompanyB, Inc. 2002. All rights reserved.
Visit us at http://example.com/companyB/myextension

About feature info

PRODUCT INFORMATION

- Primary feature controls product information
  - Splash screen
  - Window image
  - About product info
  - Initial welcome page
  - Default perspective
  - Preference default overrides

- All features can provide
  - Welcome page
  - About feature info

ECLIPSE PLATFORM - SUMMARY

- Eclipse Platform is the nucleus of IDE products
- Plug-ins, extension points, extensions
  - Open, extensible architecture
- Workspace, projects, files, folders
  - Common place to organize & store development artifacts
- Workbench, editors, views, perspectives
  - Common user presentation and UI paradigm
- Key building blocks and facilities
  - Help, team support, internationalization, …

Eclipse is a universal platform for integrating development tools

Java Development Tools

- JDT = Java development tools
- State of the art Java development environment

- Built atop Eclipse Platform
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform APIs and extension points

- Included in Eclipse Project releases
  - Available as separately installable feature
  - Part of Eclipse SDK drops

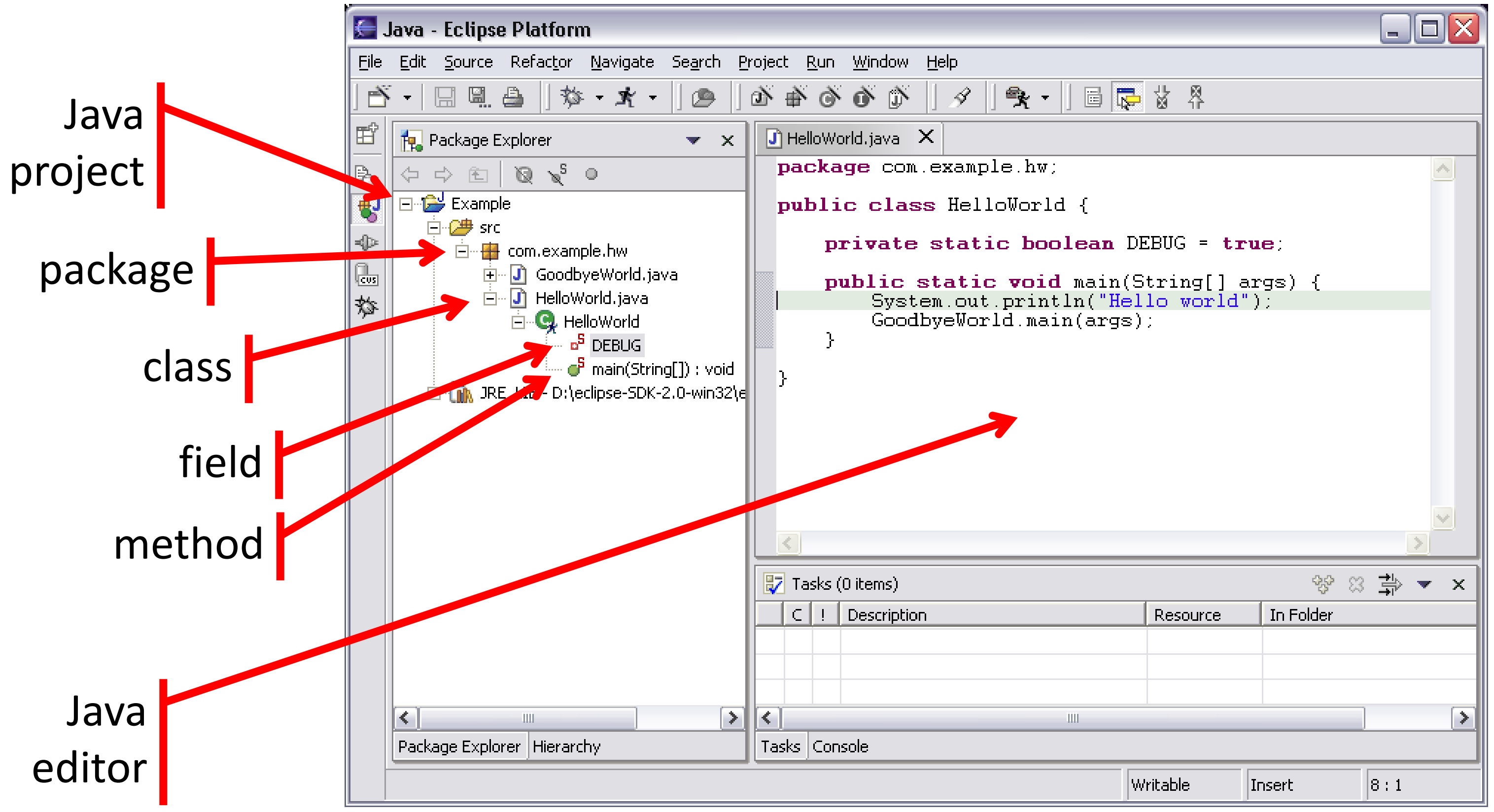
JDT Goals

- Goal: To be #1 Java IDE

- Goal: To make Java programmers smile

Java Perspective

- Java-centric view of files in Java projects
  - Java elements meaningful for Java programmers



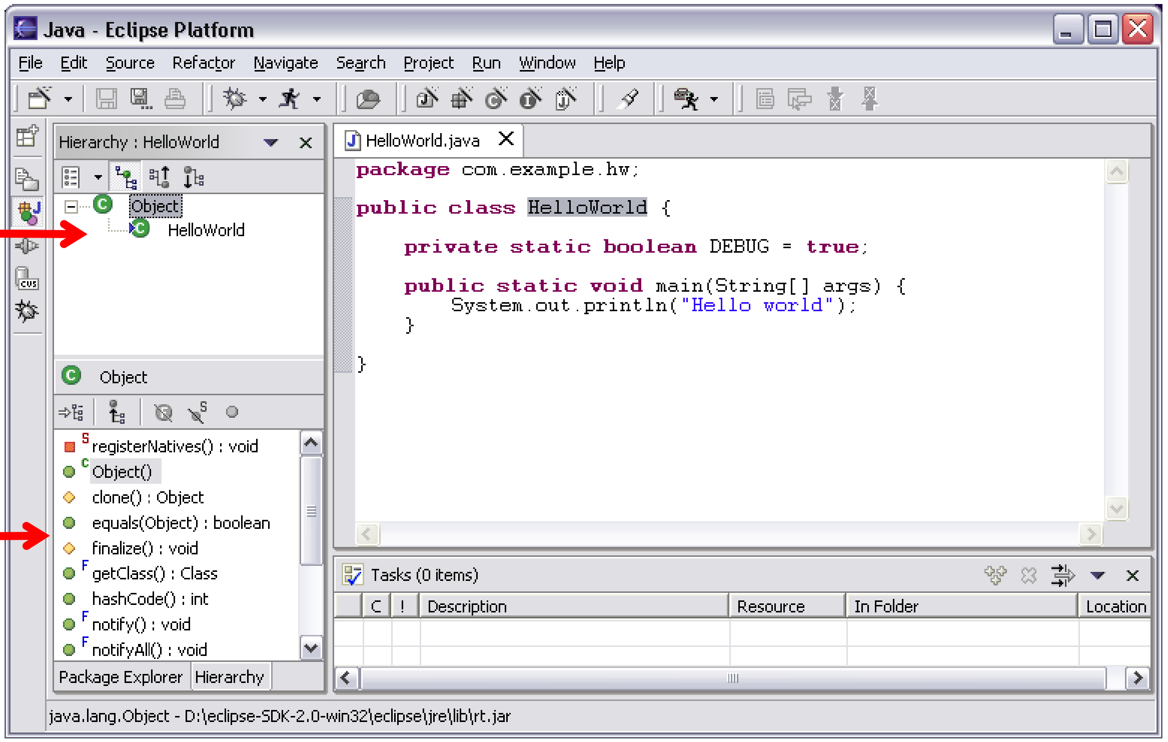
```
package com.example.hw;

public class HelloWorld {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
        GoodbyeWorld.main(args);
    }

}
```

JAVA PERSPECTIVE

- Browse type hierarchies
  - “Up” hierarchy to supertypes
  - “Down” hierarchy to subtypes

Type hierarchy

Selected type’s members


Java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Hierarchy : HelloWorld
Object
HelloWorld
HelloWorld.java
Tasks (0 items)
C ! Description Resource In Folder Location
Package Explorer Hierarchy

```
package com.example.hw;

public class HelloWorld {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

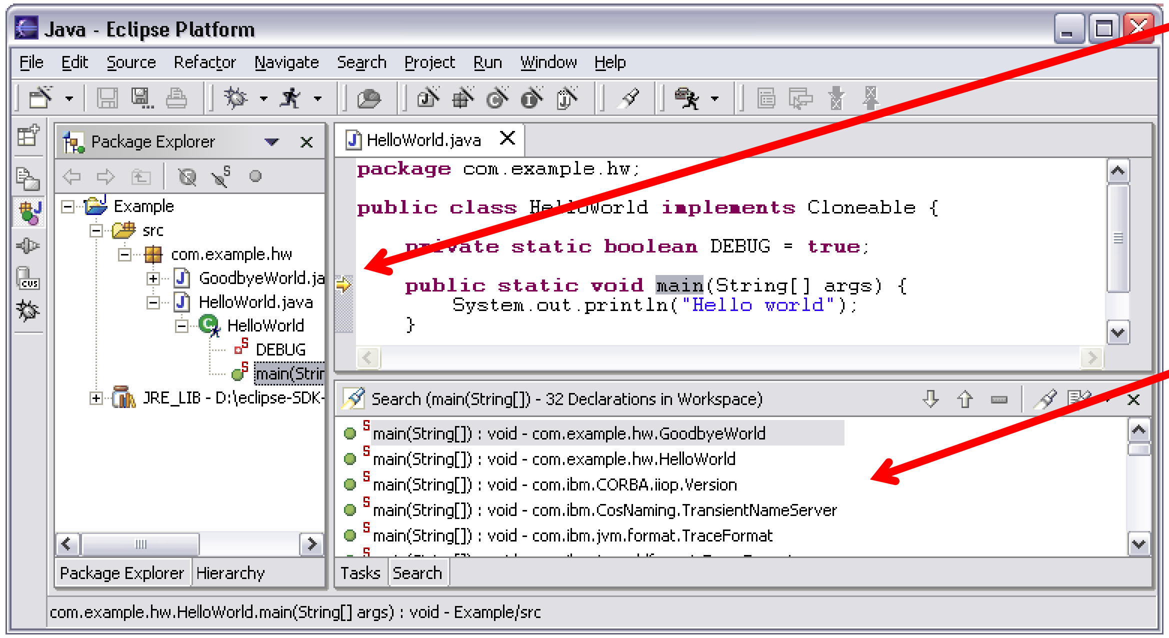
Object

```
registerNatives() : void
Object()
clone() : Object
equals(Object) : boolean
finalize() : void
getClass() : Class
hashCode() : int
notify() : void
notifyAll() : void
```

java.lang.Object - D:\eclipse-SDK-2.0-win32\eclipse\jre\lib\rt.jar

Java Perspective

- Search for Java elements
  - Declarations or references
  - Including libraries and other projects



```
package com.example.hw;

public class HelloWorld implements Cloneable {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
    }
```

Search (main(String[]) - 32 Declarations in Workspace)

```
main(String[]) : void - com.example.hw.GoodbyeWorld
main(String[]) : void - com.example.hw.HelloWorld
main(String[]) : void - com.ibm.CORBA.iiop.Version
main(String[]) : void - com.ibm.CosNaming.TransientNameServer
main(String[]) : void - com.ibm.jvm.format.TraceFormat
```

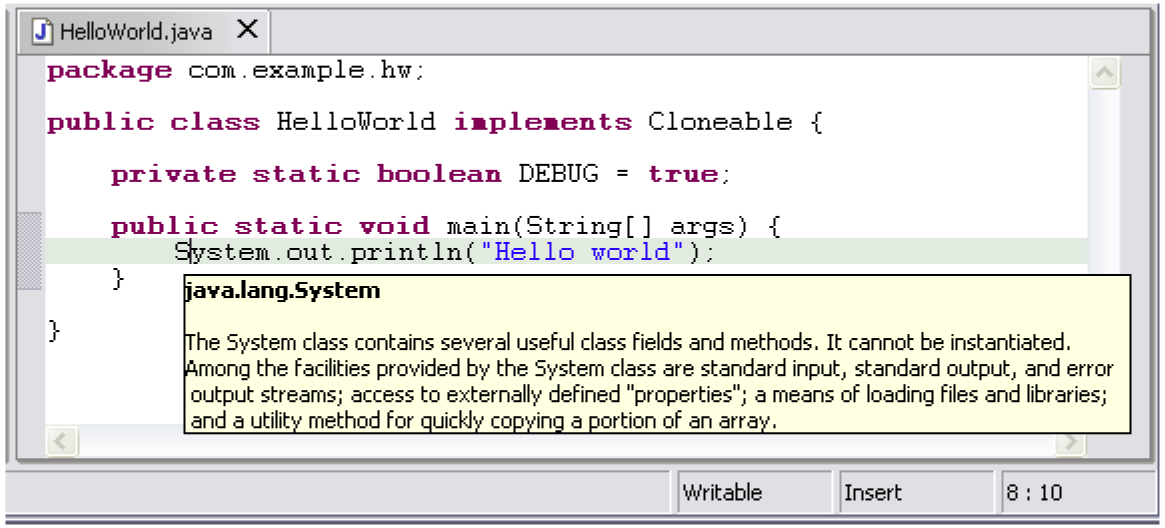
com.example.hw.HelloWorld.main(String[] args) : void - Example/src

Hits flagged in margin of editor

All search results

JAVA EDITOR

- Hovering over identifier shows Javadoc spec



```
package com.example.hw;

public class HelloWorld implements Cloneable {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

java.lang.System

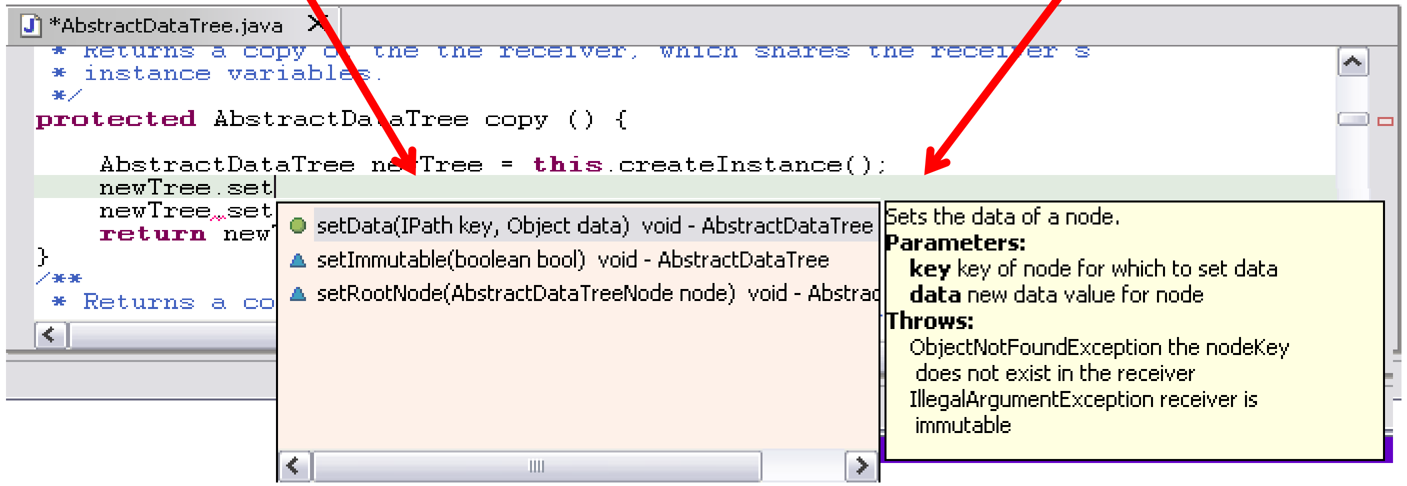
The System class contains several useful class fields and methods. It cannot be instantiated. Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined "properties"; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

Java Editor

- Method completion in Java editor

List of plausible methods

Doc for method



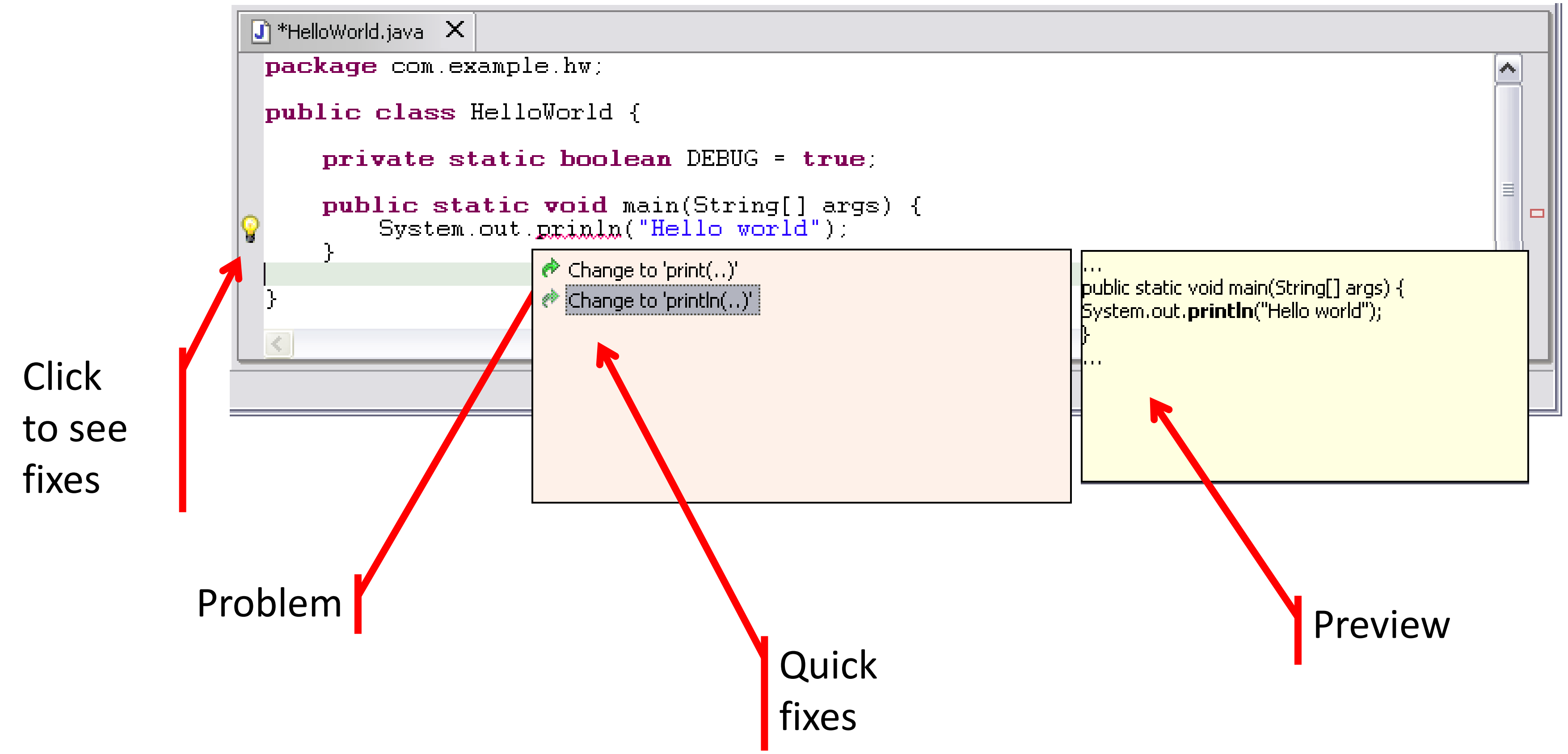
```
 * Returns a copy of the the receiver, which shares the receiver's
 * instance variables.
 */
protected AbstractDataTree copy () {

    AbstractDataTree newTree = this.createInstance();
    newTree.set
    newTree.set
    return new
}
/**
 * Returns a co
```

- setData(IPath key, Object data) void - AbstractDataTree
- setImmutable(boolean bool) void - AbstractDataTree
- setRootNode(AbstractDataTreeNode node) void - Abstrac

Sets the data of a node.
Parameters:
key key of node for which to set data
data new data value for node
Throws:
ObjectNotFoundException the nodeKey does not exist in the receiver
IllegalArgumentException receiver is immutable

JAVA EDITOR

- On-the-fly spell check catches errors early



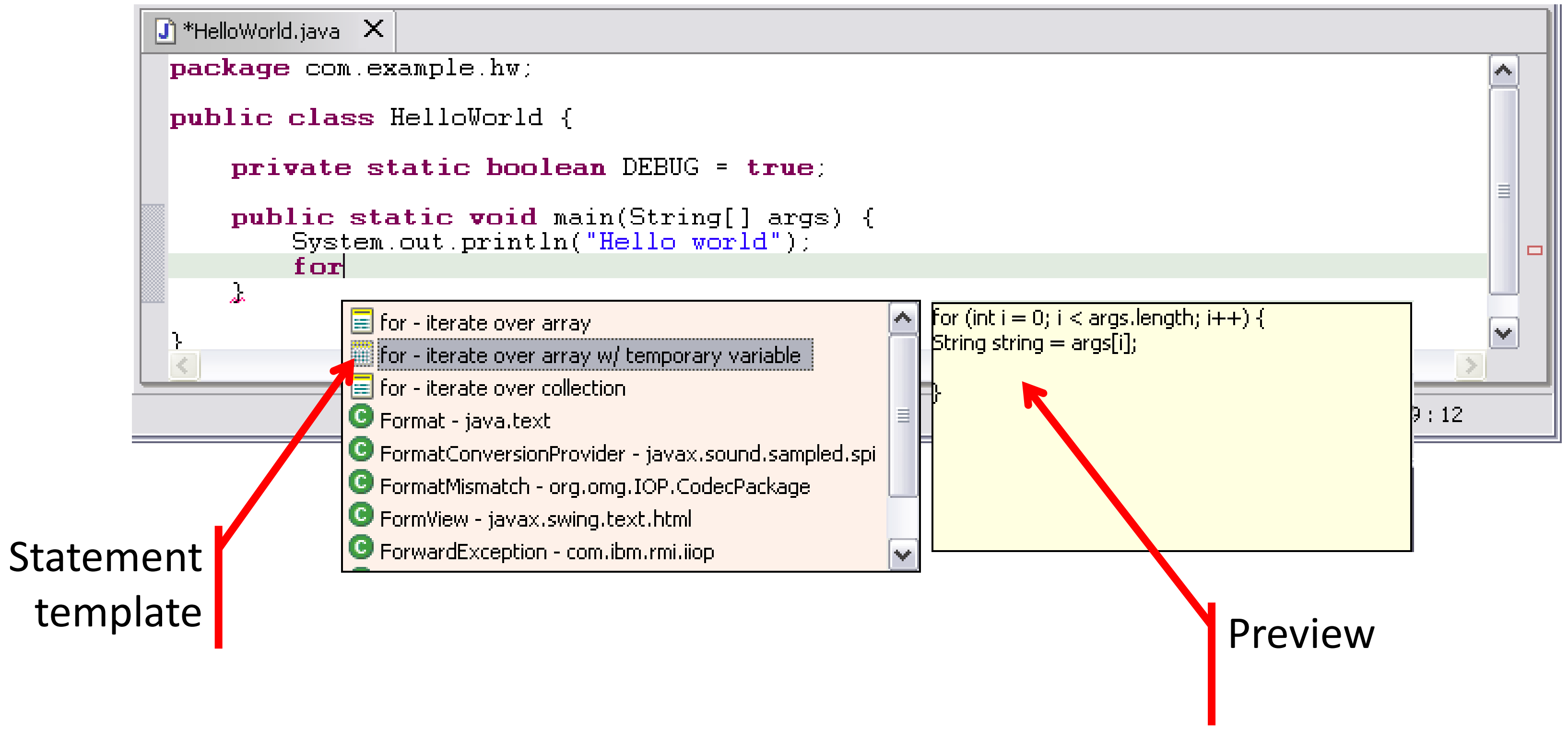
```
*HelloWorld.java
package com.example.hw;

public class HelloWorld {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.prinln("Hello world");
    }

}
```

Change to 'print(..)'
Change to 'println(..)'

```
...
public static void main(String[] args) {
System.out.println("Hello world");
}
...
```

JAVA EDITOR

- Code templates help with drudgery



```
package com.example.hw;

public class HelloWorld {

    private static boolean DEBUG = true;

    public static void main(String[] args) {
        System.out.println("Hello world");
        for
    }

}
```

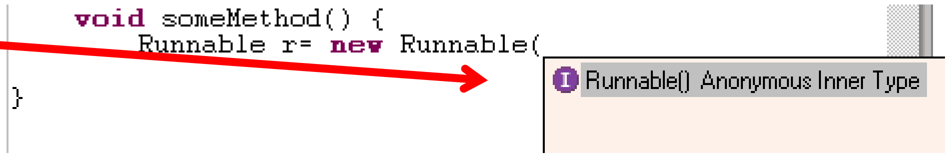
- for - iterate over array
- for - iterate over array w/ temporary variable
- for - iterate over collection
- Format - java.text
- FormatConversionProvider - javax.sound.sampled.spi
- FormatMismatch - org.omg.IOP.CodecPackage
- FormView - javax.swing.text.html
- ForwardException - com.ibm.rmi.iiop

```
for (int i = 0; i < args.length; i++) {
String string = args[i];

}
```

Java Editor

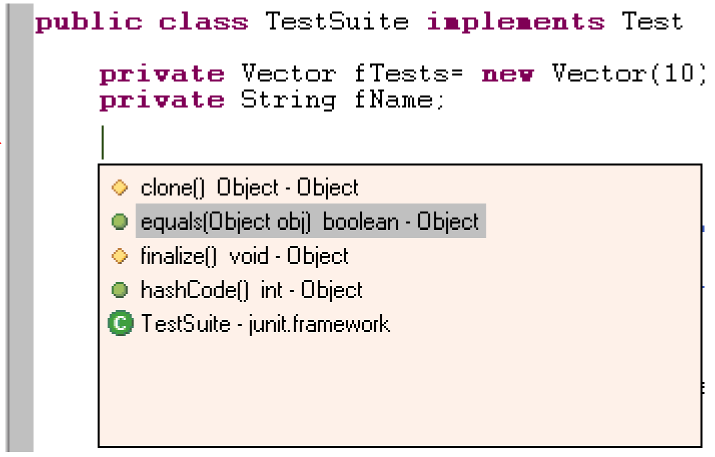
- Java editor creates stub methods

Method stub insertion for anonymous inner types



```
void someMethod() {
    Runnable r= new Runnable(
}
```

Method stub insertion for inherited methods



```
public class TestSuite implements Test

    private Vector fTests= new Vector(10)
    private String fName;
```

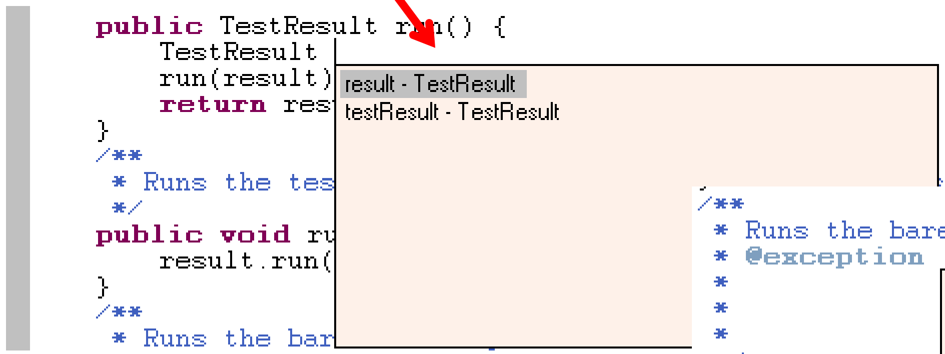
Java Editor

- Java editor helps programmers write good Java code

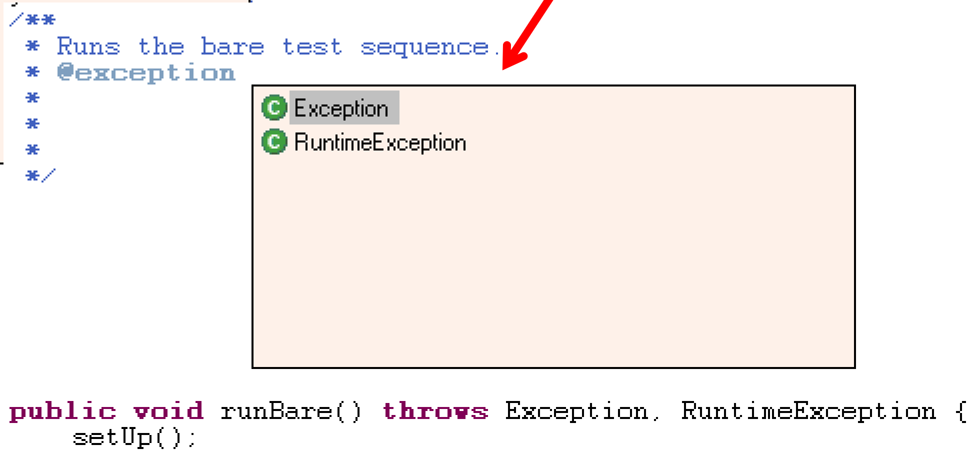
Variable name suggestion

JavaDoc code assist

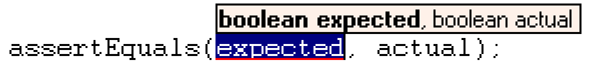
Argument hints and proposed argument names



```
public TestResult run() {
    TestResult
    run(result)
    return res
}
/**
 * Runs the tes
 */
public void ru
    result.run(
}
/**
 * Runs the bar
```



```
/**
 * Runs the bare test sequence.
 * @exception
 *
 *
 *
 */
public void runBare() throws Exception, RuntimeException {
    setUp();
```



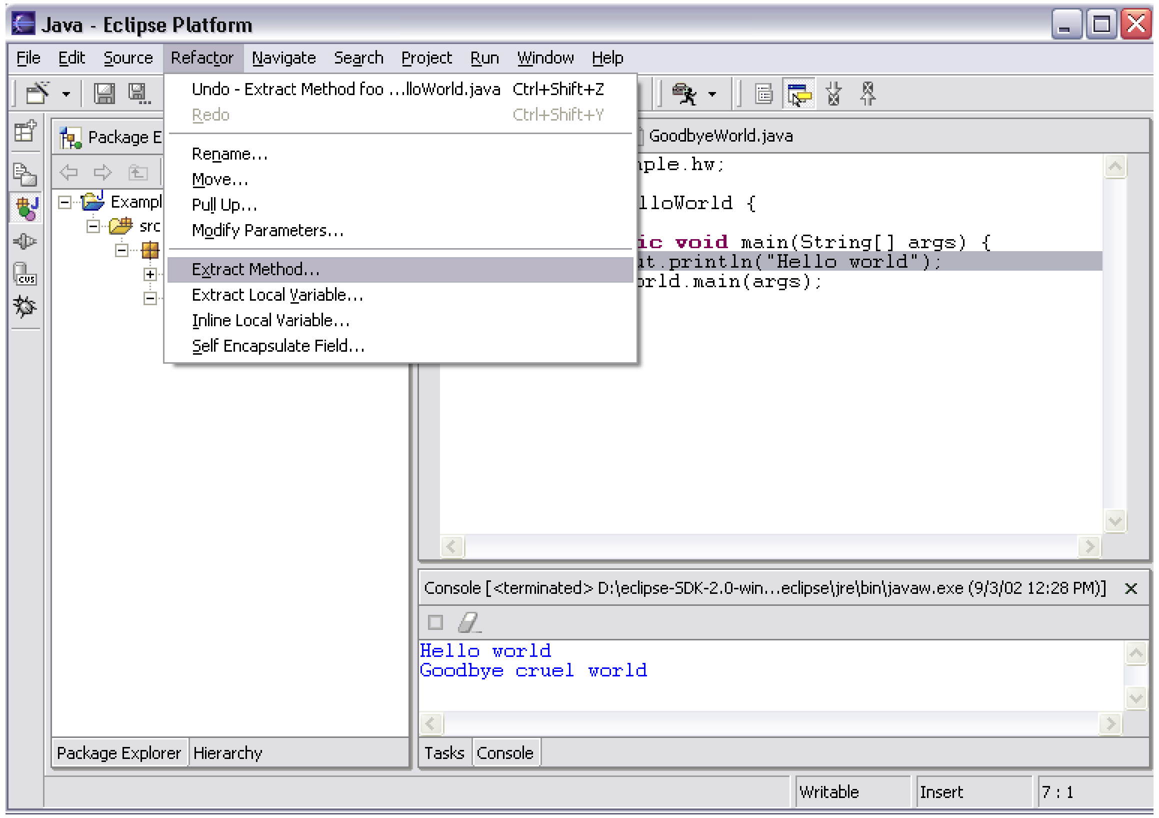
```
assertEquals(expected, actual);
```

Java Editor

- Other features of Java editor include
    - Local method history
    - Code formatter
    - Source code for binary libraries
    - Built-in refactoring

REFACTORING

- JDT has actions for refactoring Java code



```
ple.hw;

lloWorld {

ic void main(String[] args) {
ut.println("Hello world");
orld.main(args);
```

Console [<terminated> D:\eclipse-SDK-2.0-win...eclipse\jre\bin\javaw.exe (9/3/02 12:28 PM)]

```
Hello world
Goodbye cruel world
```

REFACTORING

- Refactoring actions rewrite source code
  - Within a single Java source file
  - Across multiple interrelated Java source files
- Refactoring actions preserve program semantics
  - Does not alter what program does
  - Just affects the way it does it

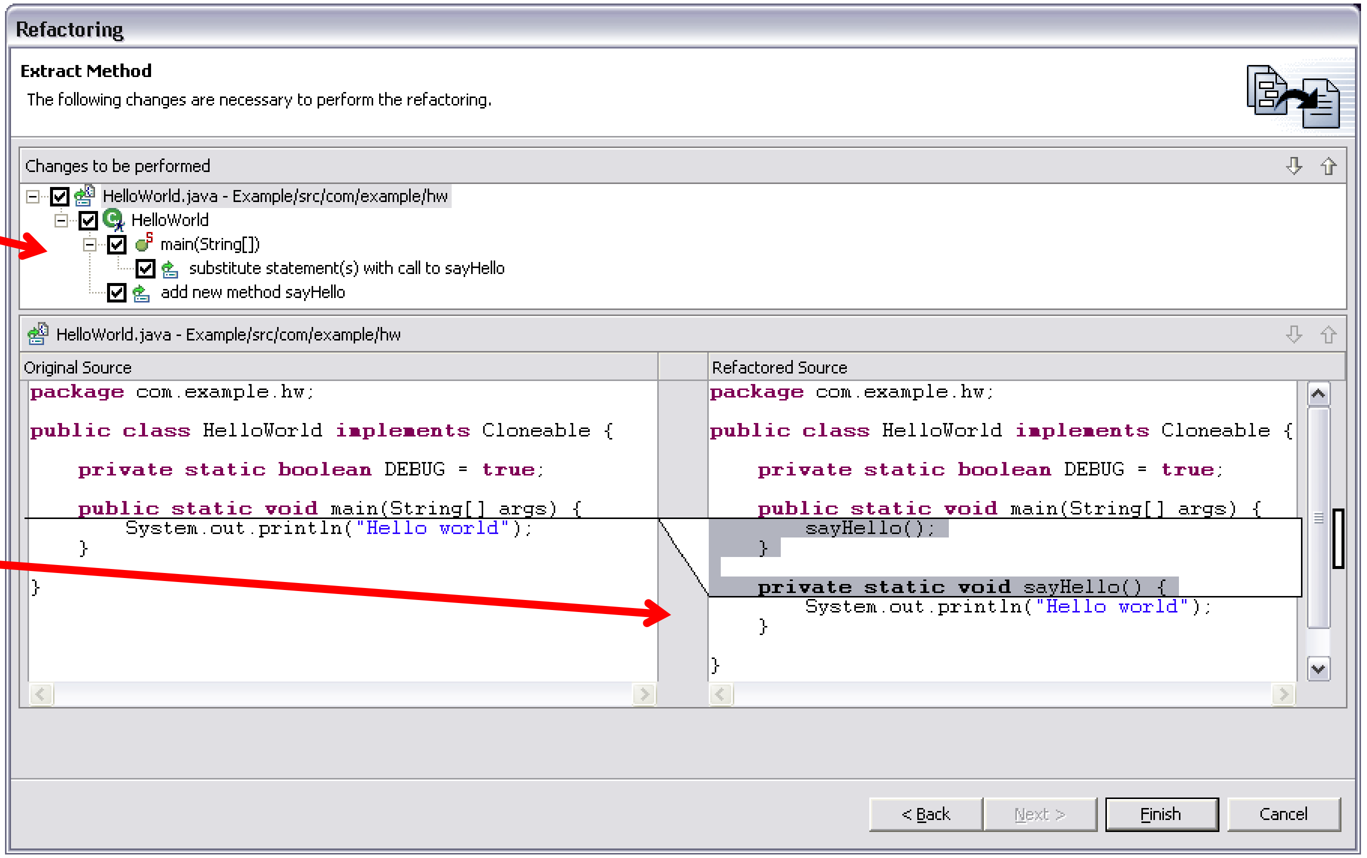
- Encourages exploratory programming
- Encourages higher code quality
  - Makes it easier to rewrite poor code

REFACTORING

- Full preview of all ensuing code changes
  - Programmer can veto individual changes

List of changes

"before" vs. "after"


Refactoring
Extract Method
The following changes are necessary to perform the refactoring.
Changes to be performed
HelloWorld.java - Example/src/com/example/hw
HelloWorld
main(String[])
substitute statement(s) with call to sayHello
add new method sayHello
HelloWorld.java - Example/src/com/example/hw
Original Source
Refactored Source
< Back
Next >
Finish
Cancel

Original Source:

```
package com.example.hw;
public class HelloWorld implements Cloneable {
    private static boolean DEBUG = true;
    public static void main(String[] args) {
        System.out.println("Hello world");
    }
}
```

Refactored Source:

```
package com.example.hw;
public class HelloWorld implements Cloneable {
    private static boolean DEBUG = true;
    public static void main(String[] args) {
        sayHello();
    }

    private static void sayHello() {
        System.out.println("Hello world");
    }
}
```

Refactoring

- Growing catalog of refactoring actions
  - Organize imports
  - Rename {field, method, class, package}
  - Move {field, method, class}
  - Extract {method, local variable, interface}
  - Inline {method, local variable}
  - Reorder method parameters
  - Push members down

  …

Eclipse Java Compiler

- Eclipse Java compiler
  - JCK-compliant Java compiler (selectable 1.3 and 1.4)
  - Helpful error messages
  - Generates runnable code even in presence of errors
  - Fully-automatic incremental recompilation
  - High performance
  - Scales to large projects
- Multiple other uses besides the obvious
  - Syntax and spell checking
  - Analyze structure inside Java source file
  - Name resolution
  - Content assist
  - Refactoring
  - Searches

ECLIPSE JAVA DEBUGGER

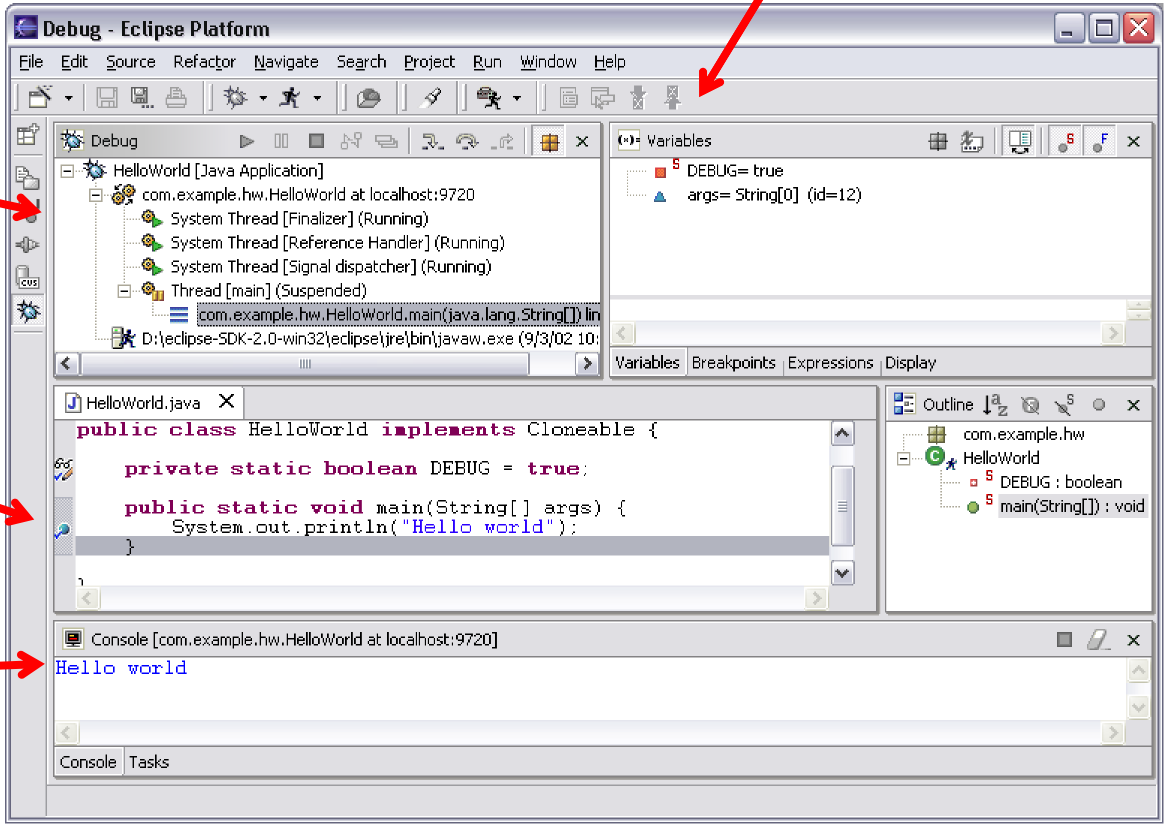
- Run or debug Java programs

Local variables

Threads and stack frames

Editor with breakpoint marks

Console I/O


Debug - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
HelloWorld.java
Outline
Console [com.example.hw.HelloWorld at localhost:9720]
Console Tasks

```
HelloWorld [Java Application]
  com.example.hw.HelloWorld at localhost:9720
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    System Thread [Signal dispatcher] (Running)
    Thread [main] (Suspended)
      com.example.hw.HelloWorld.main(java.lang.String[]) lin
  D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (9/3/02 10:
```

```
DEBUG= true
args= String[0]  (id=12)
```

```java
public class HelloWorld implements Cloneable {

   private static boolean DEBUG = true;

   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

```
com.example.hw
HelloWorld
  DEBUG : boolean
  main(String[]) : void
```

```
Hello world
```

Eclipse Java Debugger

- Run Java programs
  - In separate target JVM (user selectable)
  - Console provides stdout, stdin, stderr
  - Scrapbook pages for executing Java code snippets
- Debug Java programs
  - Full source code debugging
  - Any JPDA-compliant JVM
- Debugger features include
  - Method and exception breakpoints
  - Conditional breakpoints
  - Watchpoints
  - Step over, into, return; run to line
  - Inspect and modify fields and local variables
  - Evaluate snippets in context of method
  - Hot swap (if target JVM supports)

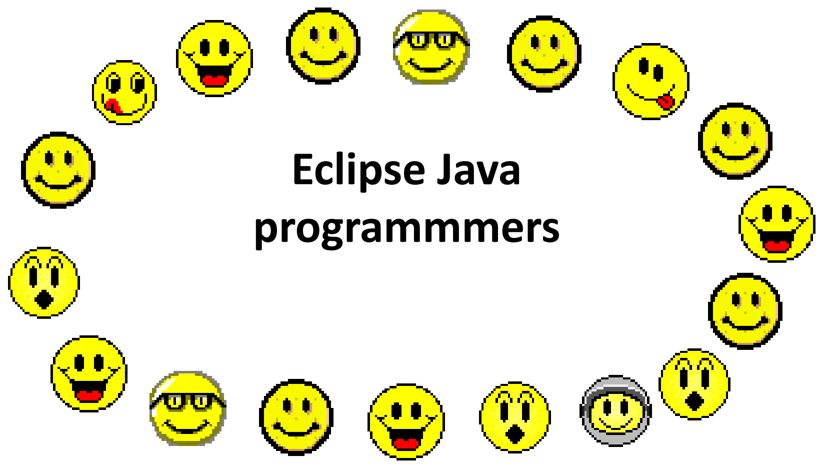
JDT APIs

- JDT APIs export functionality to other plug-ins

- Java model
  - Java-centric analog of workspace
  - Tree of Java elements (down to individual methods)
  - Java element deltas
  - Type hierarchies
  - Model accurate independent of builds
- Building blocks
  - Java scanner
  - Java class file reader
  - Java abstract syntax trees (down to expressions)
- Many others…

ECLIPSE JDT - SUMMARY

- JDT is a state of the art Java IDE
- Java views, editor, refactoring
  - Helps programmer write and maintain Java code
- Java compiler
  - Takes care of translating Java sources to binaries
- Java debugger
  - Allows programmer to get inside the running program



PLUG-IN DEVELOPMENT ENVIRONMENT

- PDE = Plug-in development environment
- Specialized tools for developing Eclipse plug-ins

- Built atop Eclipse Platform and JDT
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform and JDT APIs and extension points

- Included in Eclipse Project releases
  - Separately installable feature
  - Part of Eclipse SDK drops

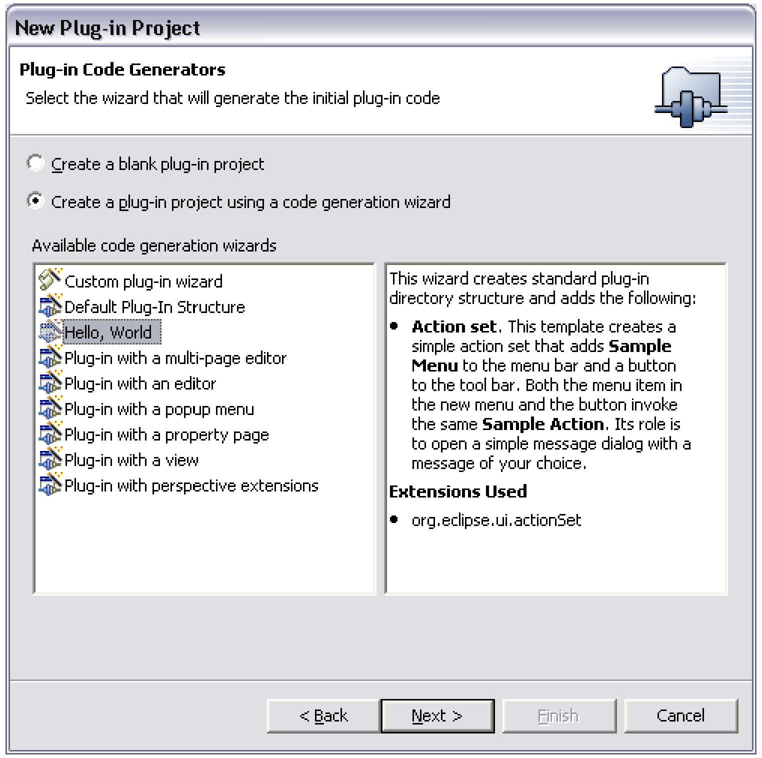
PDE Goals

- Goal: To make it easier to develop Eclipse plug-ins

- Goal: Support self-hosted Eclipse development

PDE

- PDE templates for creating simple plug-in projects



New Plug-in Project

Plug-in Code Generators

Select the wizard that will generate the initial plug-in code

( ) Create a blank plug-in project

(●) Create a plug-in project using a code generation wizard

Available code generation wizards

- Custom plug-in wizard
- Default Plug-In Structure
- Hello, World
- Plug-in with a multi-page editor
- Plug-in with an editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with perspective extensions

This wizard creates standard plug-in directory structure and adds the following:

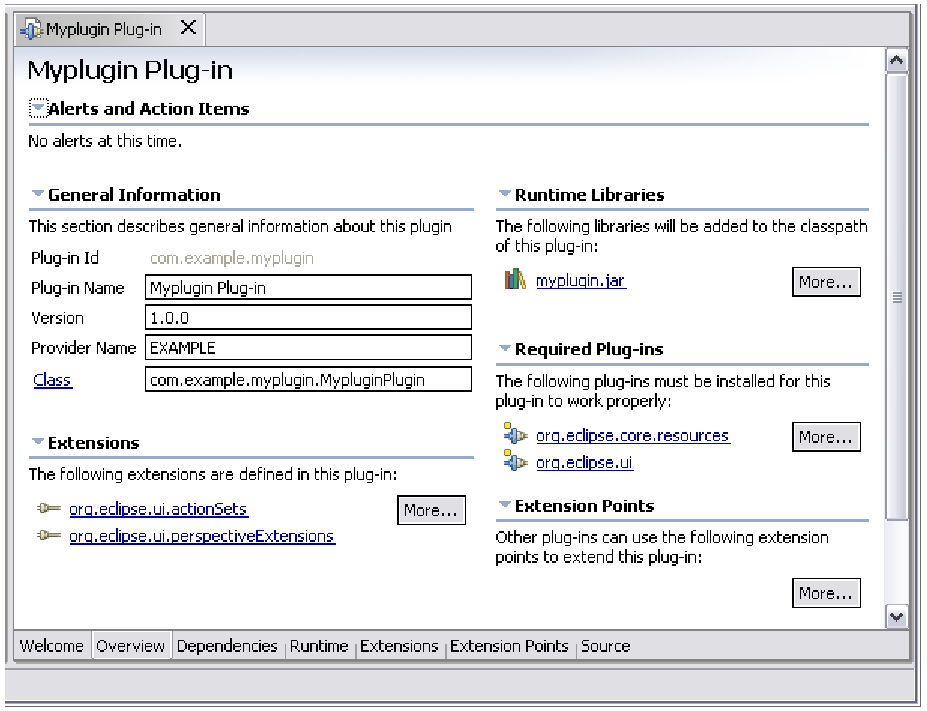
- **Action set**. This template creates a simple action set that adds **Sample Menu** to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same **Sample Action**. Its role is to open a simple message dialog with a message of your choice.

Extensions Used

- org.eclipse.ui.actionSet

PDE

- Specialized PDE editor for plug-in manifest files



Myplugin Plug-in

Myplugin Plug-in

Alerts and Action Items

No alerts at this time.

General Information

This section describes general information about this plugin

Plug-in Id	com.example.myplugin
Plug-in Name	Myplugin Plug-in
Version	1.0.0
Provider Name	EXAMPLE
Class	com.example.myplugin.MypluginPlugin

Extensions

The following extensions are defined in this plug-in:

org.eclipse.ui.actionSets
org.eclipse.ui.perspectiveExtensions

Runtime Libraries

The following libraries will be added to the classpath of this plug-in:

myplugin.jar

Required Plug-ins

The following plug-ins must be installed for this plug-in to work properly:

org.eclipse.core.resources
org.eclipse.ui

Extension Points

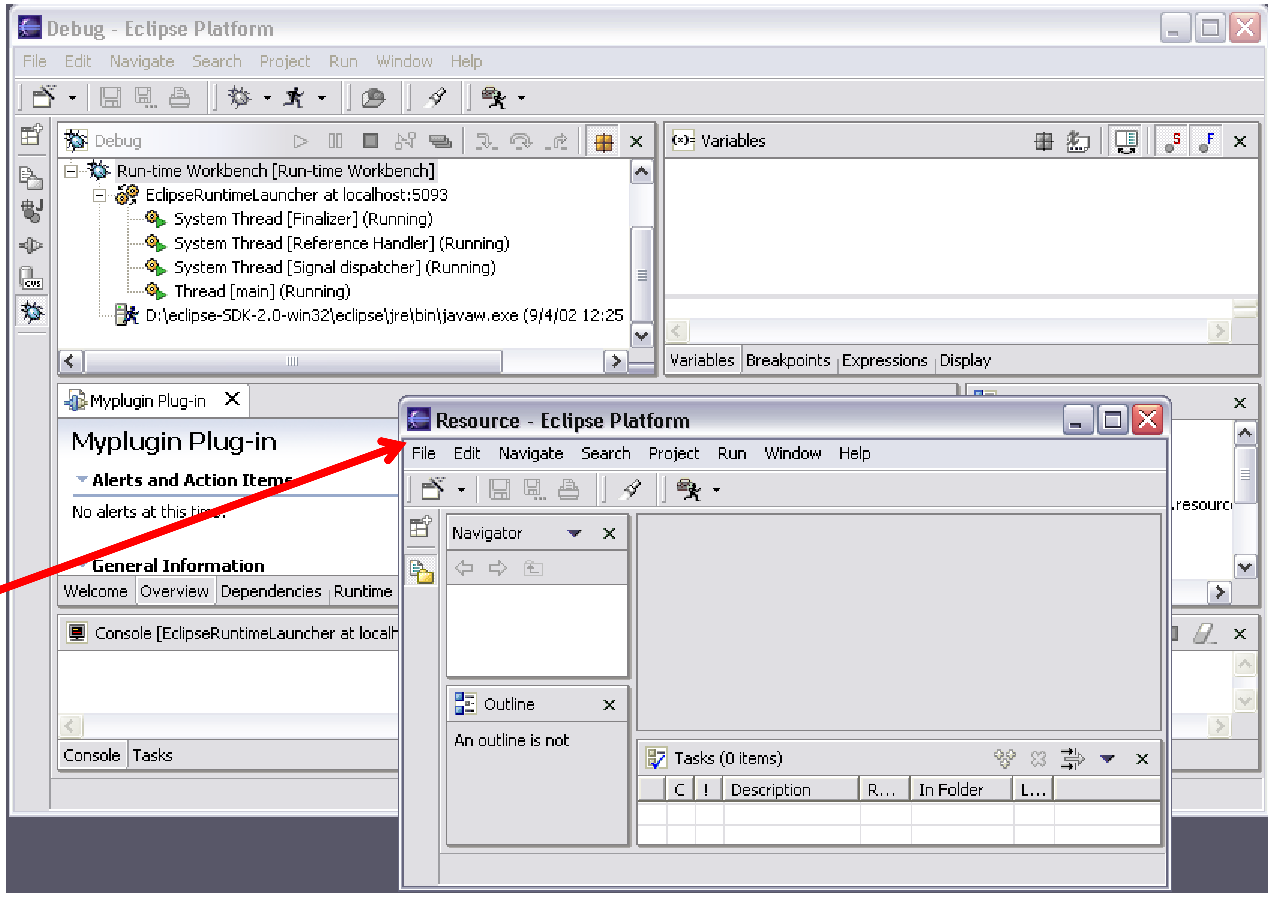
Other plug-ins can use the following extension points to extend this plug-in:

PDE

- PDE runs and debugs another Eclipse workbench

1. Workbench running PDE (host)

2. Run-time workbench (target)



```
Run-time Workbench [Run-time Workbench]
  EclipseRuntimeLauncher at localhost:5093
    System Thread [Finalizer] (Running)
    System Thread [Reference Handler] (Running)
    System Thread [Signal dispatcher] (Running)
    Thread [main] (Running)
  D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (9/4/02 12:25
```

Debug - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
Myplugin Plug-in
Myplugin Plug-in
Alerts and Action Items
No alerts at this time.
General Information
Welcome Overview Dependencies Runtime
Console [EclipseRuntimeLauncher at localh
Console Tasks
Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
An outline is not
Tasks (0 items)
C ! Description R... In Folder L...
.resourc

PDE - Summary

- PDE makes it easier to develop Eclipse plug-ins
- PDE also generates Ant build scripts
  - Compile and create deployed form of plug-in

PDE is basis for self-hosted Eclipse development

Eclipse Operating Environments

- Eclipse Platform currently* runs on
  - Microsoft® Windows® XP, 2000, NT, ME, 98SE
  - Linux® on Intel x86 - Motif, GTK
    - RedHat Linux 8.0 x86
    - SuSE Linux 8.1 x86
  - Sun Solaris 8 SPARC – Motif
  - HP-UX 11i hp9000 – Motif
  - IBM® AIX 5.1 on PowerPC – Motif
  - Apple Mac OS® X 10.2 on PowerPC – Carbon
  - QNX® Neutrino® RTOS 6.2.1 - Photon®

Other Operating Environments

- Most Eclipse plug-ins are 100% pure Java
  - Freely port to new operating environment
  - Java2 and Eclipse APIs insulate plug-in from OS and window system

- Gating factor: porting SWT to native window system
- Just added in 2.1*
  - Mac OS X PowerPC – Carbon window system
  - QNX Neutrino RTOS Intel x86 - Photon window system

- Eclipse Platform also runs “headless”
  - Example: help engine running on server

Who's Shipping on Eclipse?

- Commercial products*
  - 10 Technology – Visual PAD
  - Assisi – V4ALL Assisi GUI-Builder
  - Bocaloco – XMLBuddy
  - Borland – Together Edition for WebSphere Studio
  - Catalyst Systems – Openmake
  - Computer Associates – AllFusion Harvest Change Manager VCM
  - Ensemble Systems – Glider for Eclipse
  - Fujitsu – Interstage
  - Genuitec – EASIE Plug-ins
  - HP – OpenCall Media Platform OClet Development Environment
  - James Holmes – Struts Console
  - Instantiations – CodePro Studio

Who's Shipping on Eclipse?

- IBM uses Eclipse for
  - WebSphere® Studio Family
    - WebSphere Studio Homepage Builder
    - WebSphere Studio Site Developer (WSSD)
    - WebSphere Studio Application Developer (WSAD)
    - WebSphere Studio Application Developer Integration Edition (WSADIE)
    - WebSphere Studio Enterprise Developer (WSED)
    - WebSphere Studio Device Developer (WSDD)
    - WebSphere Development Studio for iSeries
  - Rational® XDE Professional: Java Platform Edition
  - Tivoli Monitoring Workbench

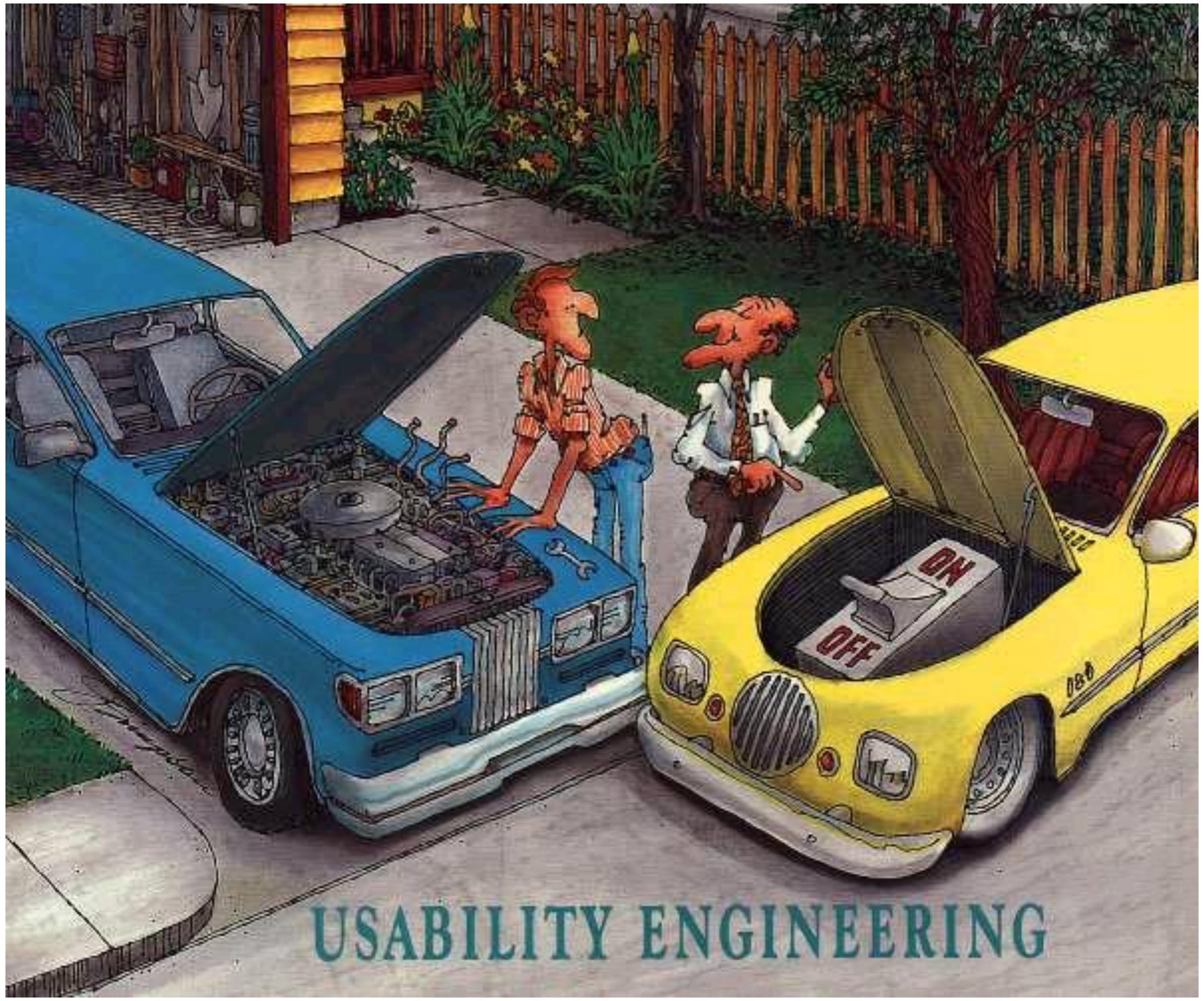
Who's Shipping on Eclipse?

- Commercial products*
  - Interwoven – TeamSite repository
  - Intland – CodeBeamer
  - LegacyJ – PERCobol
  - Merant – PVCS Version Manager
  - MKS – Source Integrity Enterprise plug-in
  - Mobile Media – Grand-Rapid Browser
  - mvmsoft – Slime UML
  - No Magic Inc. – MagicDraw UML
  - Object Edge – Weblogic Plug-in
  - ObjectLearn – Lomboz
  - Omondo – EclipseUML
  - Ontogenics – hyperModel

Who's Shipping on Eclipse?

- Commercial products*
  - Parasoft – Jtest
  - ProSyst – Eclipse OSGi Plug-in
  - QNX – QNX Momentics
  - Quest Software – JProbe integration
  - Serena Software – ChangeMan DS
  - SlickEdit – Visual SlickEdit Plug-in
  - Systinet – WASP Developer
  - THOUGHT – CocoBase Enterprise O/R
  - TimeSys – TimeStorm 2.0
  - xored – WebStudio IDE for PHP

9) PRINCIPLE FOR USABLE DESIGN & USABILITY METRICS


ON
OFF
USABILITY ENGINEERING

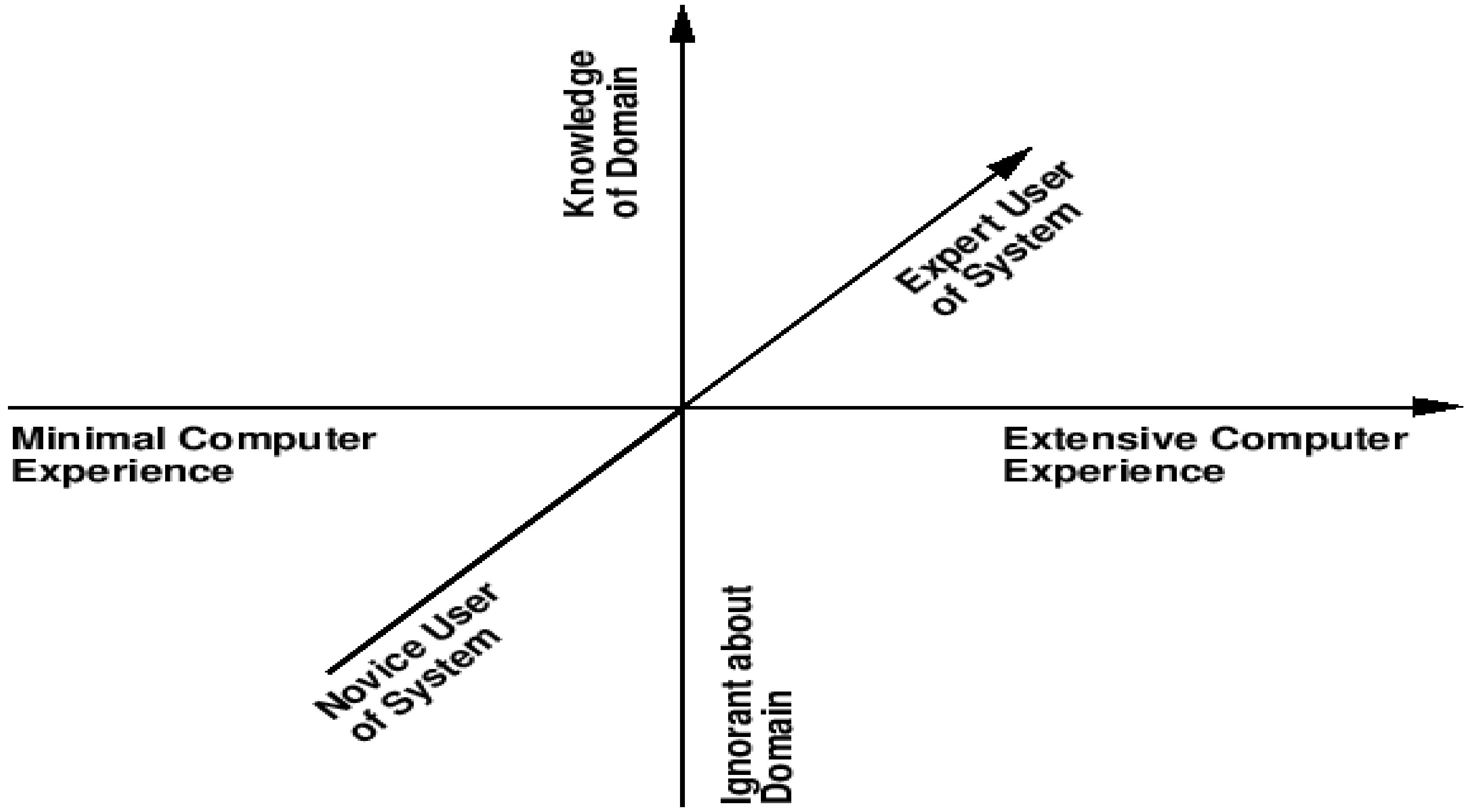
USABILITY ENGINEERING LIFECYCLE

1. Know the User
2. Competitive Analysis
3. Set Usability Goals
4. Parallel Design
5. Participatory Design
6. Co-ordinated Design of Total Interface
7. Applying Guidelines
8. Prototyping
9. Usability Evaluation (Inspection and Testing)
10. Iterative Design
11. Follow-up Studies

1. Know the User

- **Observe Users in Working Environment**:
  Site visits, unobtrusive observation. Don't believe their superiors!
- **Individual User Characteristics**:
  Classify users by experience (see next slide), educational level, age, amount of prior training, etc.
- **Task Analysis**:
  Users' overall goals, *current* approach, model of task, prerequisite information, exceptions to normal work flow.
- **Functional Analysis**:
  Functional reason for task: what really needs to be done, and what are merely surface procedures?
- **Evolution of User and Task**:
  Users change as they use system, then use system in new ways.
  - Draw up set of typical *user scenarios* to set the scene for brainstorming, thinking aloud, or cognitive walkthrough, etc.

Categories of User Experience



2. Competitive Analysis

- Competitive analysis of software components:
  - Use existing interface framework as far as possible (Motif, MS-Windows, Java AWT) - saves a *lot* of work.
  - Use existing components and applications rather than re-inventing the wheel.
- Competitive analysis of competing systems:
  - Analyse competing products heuristically or empirically.
  - “Intelligent borrowing” of ideas from other systems.

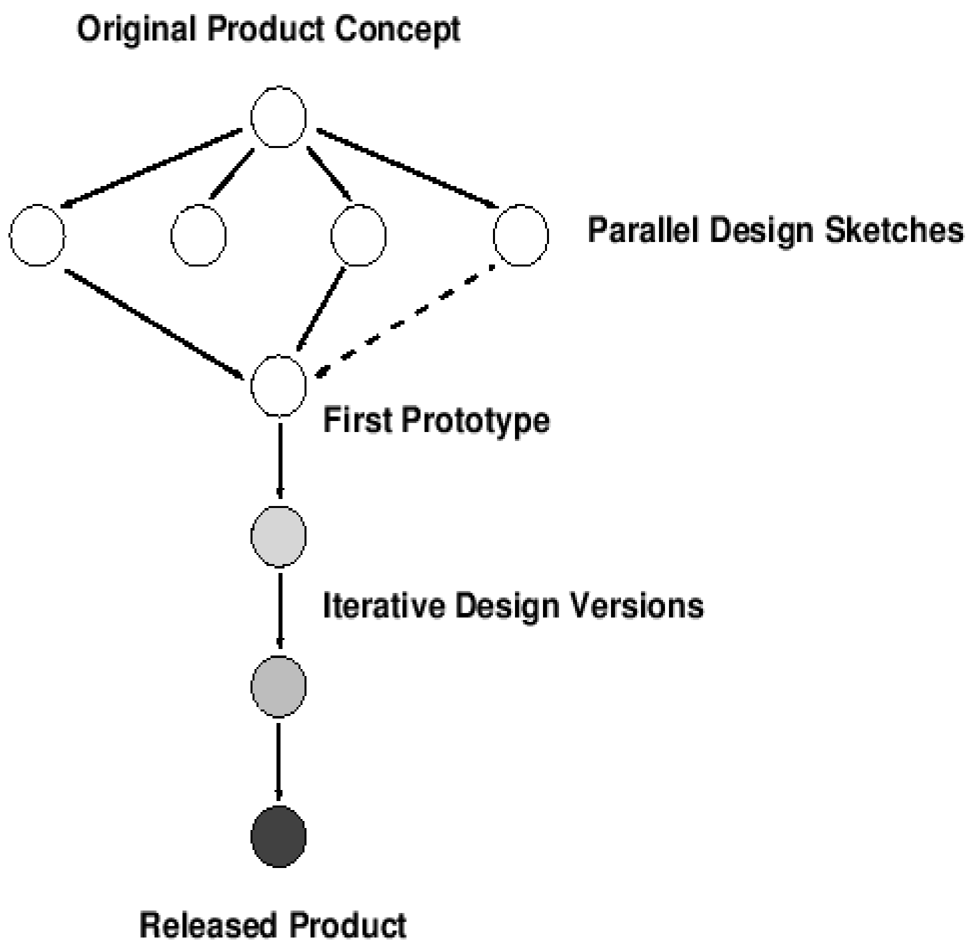
3. Set Usability Goals

- Decide in advance on usability metrics and desired level of *measured* usability (usability goal line).
- Financial impact analysis - estimate savings based on *loaded cost* of users, compared to cost of usability effort.

Errors per Hour			
Optimal	Target	Current	Unacceptable
0	1=3	4.5	>5

4. Parallel Design

- Explore design alternatives - designers should work *independently*, then compare draft designs.
- Brainstorm with whole team (engineers, graphic designer, writer, marketing types, usability specialist, one or two representative users).



How to Brainstorm

- Meet away from usual workplace (different building, hut in the mountains).
- Use plenty of paper. Cover the walls with it!
- Draw. Scribble. Use lots of coloured pens.
- Three rules during brainstorming:
  - No one is allowed to criticise another's ideas.
  - Engineers must not say it can't be implemented.
  - Graphic designer must not laugh at engineers' drawings.
- Only *after* brainstorming, organise ideas and consider their practicality and viability.

5. Participatory Design

- Have access to pool of *representative* users.
- Users become first class members in design process
  - active collaborators vs passive participants
- Users considered subject matter experts
  - know all about the work context
- Iterative process
  - all design stages subject to revision
- Guided discussion of prototypes, paper mock-ups, screen designs with representative users.
- Similar to JAD though of British/Scandinavian origin

5. Participatory Design (contd)

- Users are excellent at reacting to suggested system designs
  - Designs must be concrete and visible
- Users bring in important “folk” knowledge of work context
  - Knowledge may be otherwise inaccessible to design team
- Greater buy-in for the system often results

- Hard to get a good pool of end users
  - Expensive, reluctance ...
- Users are not expert designers
  - Don’t expect them to come up with design ideas from scratch
- The user is not always right
  - Don’t expect them to know what they want

Methods of Involving Users

- At the very least, talk to users
    - Surprising how many designers don’t!
- Interviews
    - Used to discover user’s culture, requirements, expectations, etc.
- Contextual inquiry:
    - Interview users in their workplace, as they are doing their job
- Explain designs
    - Describe what you’re going to do
    - Get input at all design stages
        - all designs subject to revision
- Important to have visuals and/or demos
    - People react far differently with verbal explanations

6. Coordinated Design of Total Interface

- Consistency across *total* interface: documentation, online help, tutorials, videotapes, training classes as well as screens and dialogues.
  By means of:
- *Interface standards*: specific rules as to how interface should look and feel.
- *Widget libraries*: shared code implementing standard UI functionality.
- *Shared culture*: training, meetings, “interface evangelist”

7. Applying Guidelines

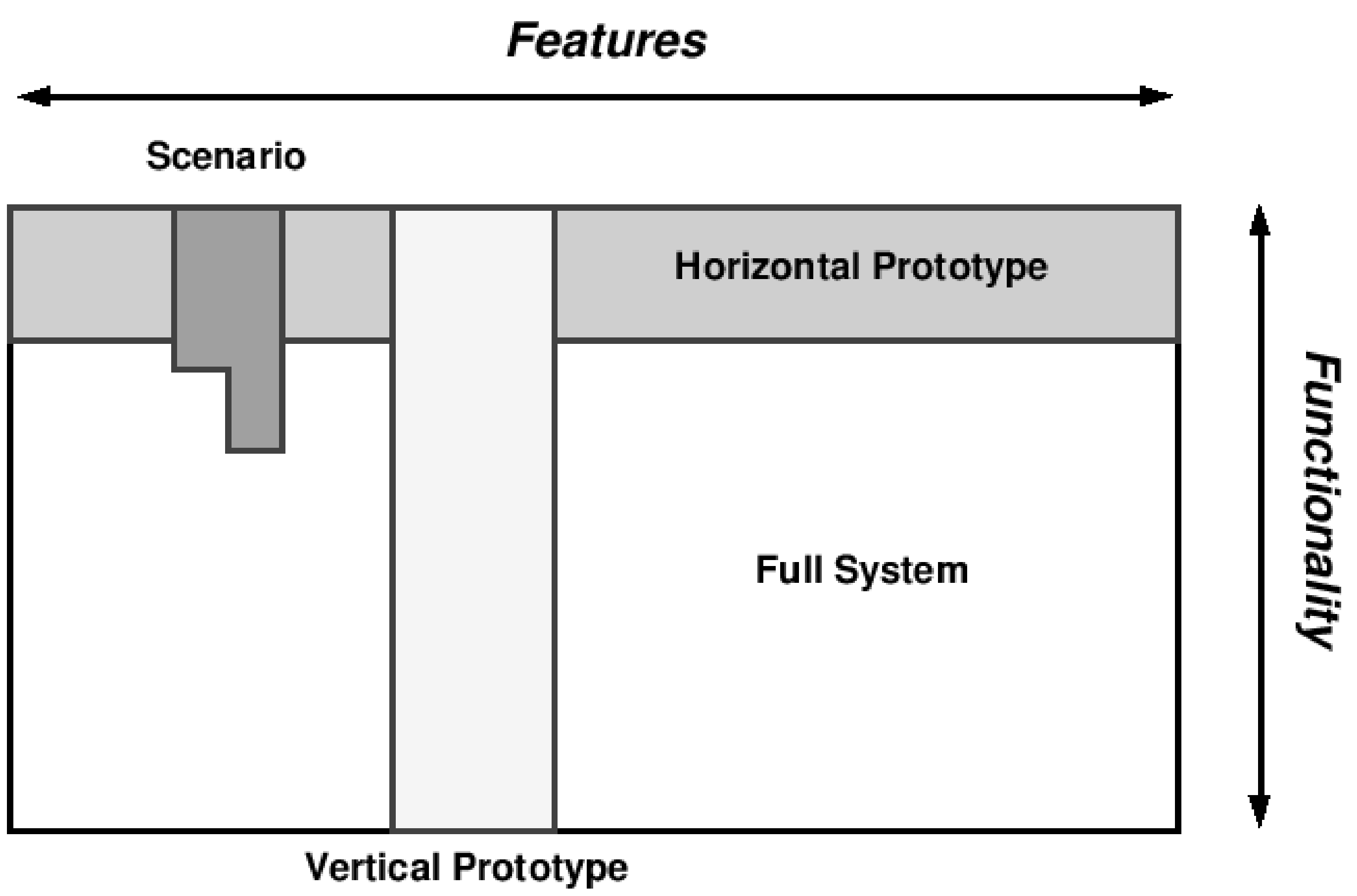
- *Guidelines* ...general principles and advice about usability characteristics of interfaces:
  - Smith S. & Mosier J.; Design Guidelines for Designing User Interface Software; The MITRE Corp., 1986. [944 guidelines] ftp://ftp.cis.ohio-state.edu/pub/hci/Guidelines
  - Brown C.; *Human-Computer Interface Design Guidelines*; Ablex, NJ, 1988. [302 guidelines]
  - Mayhew D.; *Principles and Guidelines in Software User Interface Design*; Prentice-Hall, 1991. [288 guidelines]
- Can be intimidating - often hundreds or thousands of specific recommendations.
- Many conflicting guidelines
- Impractical for rapid design.

8. Prototyping

- Perform usability evaluation as *early* as possible in the design cycle by building and evaluating prototypes.
- Prototypes cut down on either the number of features, or the depth of functionality of features:
  - *Vertical Prototype*: in-depth functionality for a few selected features.
  - *Horizontal Prototype*: full interface features, but no underlying functionality.
  - *Scenario*: only features and functionality along a pre-specified scenario (task) or path through the interface.

    "Description of an individual user using specific computer facilities to achieve a specific outcome under specified circumstances along a certain time dimension."

DIMENSIONS OF PROTOTYPING



Prototyping

Early Design

Brainstorm different representations Choose a representation Rough out interface style Task centered walkthrough and redesign	Low fidelity paper prototypes
Fine tune interface, screen design Heuristic evaluation and redesign	Medium fidelity prototypes
Usability testing and redesign	High fidelity prototypes / restricted systems
Limited field testing	
Alpha/Beta tests	Working systems

Late Design

9. Usability Evaluation

- **Usability Inspection**
  Inspection of interface design using *heuristics* and *judgement* (no user tests).
  - Heuristic evaluation
  - Cognitive walkthroughs

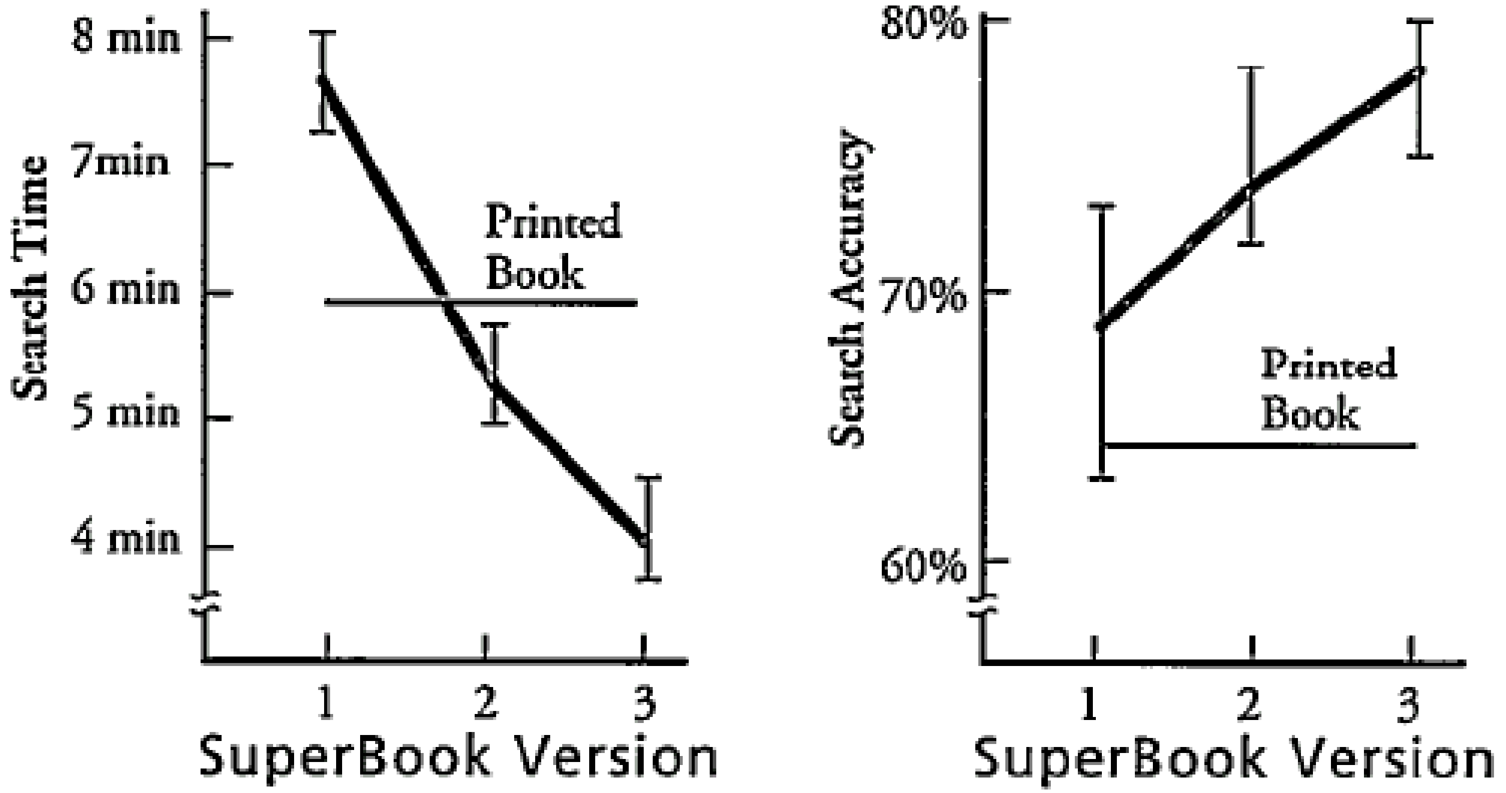
- **Usability Testing**
  Empirical testing of interface design with real users.
  - Paper and pencil tests
  - Formal experiments
  - Thinking aloud protocol analysis
  - Query techniques

10. Iterative Design

- *Severity ratings* of usability problems discovered via empirical testing.
- Fix problems in new iteration of interface.
- Capture *design rationale*: record reasons why changes were made. (gIBIS[1]) (Carroll & Rosson[2])
- Evaluate new version of interface.
- Design, test, redesign.

[1]See Dix et al 2nd Ed. p.214
[2]Psychological Design Rationale

Benefits of Iterative Design



Iterative Design Improvement with SuperBook. (Egan et al.) ACM Trans. Information Systems 7(1), Jan., 1989 pp. 30-57.

11. Follow-up Studies

- Important usability data can be gathered after the release of a product for the next version:
- Specific field studies (interviews, questionnaires, observation).
- Standard marketing studies.
- Instrumented versions of software log data.
- Analyse user complaints, modification requests, bug reports. (Therac-25)

Planning Usability Activities

- Prioritise activities.
- Write down explicit plan for each activity.
- Subject plan to independent review (e.g. colleague from different project).
- Perform pilot activity with about 10% of total resources, then revise plan for remaining 90%. [*Always* perform a pilot study!]

Survey of the usability budgets of 31 projects having some usability activities.

	Q1	Median	Q3
Project size (person-years)	11	23	58
Actual usability budget (% total)	4	6	15
Ideal usability budget (% total)	6	10	21
Actual usability effort (person-years)	1	1.5	2
Ideal usability effort (person-years)	1.7	2.3	3.8

SUMMARY

- User-centered design is different than traditional methodologies
  - Iterative prototype + evaluate
  - Prevents problems up front
  - Know the user and involve them in design