

# SURF Tutorial

**Bootstrapping a pattern.** SURF presents examples that have been indicated to be positive (indicated with a **green** header, e.g. Example 1024) or negative (**red** header, e.g. Example 1001). Our goal is to find more examples resembling the **positive** examples as we interactively provide feedback. The details of the API (e.g., its semantics, why is a given usage is misuse, and the consequences of their misuse) **can be ignored**.

These are the examples indicated as positive and negative. The specific details of the code examples can be ignored. Scroll down and click on the Infer Pattern button at the bottom of the page to view a summary, then refine the pattern.

Positive

Example 1024:

```
private static byte[] computeSha256Digest(byte[] input) {
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
        messageDigest.update(input);
        return messageDigest.digest();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

Example 1001:

Negative

```
public final static String getMessageDigest(byte[] buffer) {
    char[] hexDigits = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c',
        'd', 'e', 'f'};
    try {
        MessageDigest mdTemp = MessageDigest.getInstance("MD5");
        mdTemp.update(buffer);
        byte[] md = mdTemp.digest();
    }
}
```

**Scroll** to the bottom of the page and **click** on “Infer Pattern” to find the most general pattern that matches the positive examples. Next, you can **inspect** the most general pattern that

matches the positive examples in the left pane, <sup>1</sup>, and provide feedback on the features in

the right panel, <sup>2</sup>. To **provide feedback** to specialize the pattern, **check the suggest checkboxes**. A checkmark means that SURF should consider including this feature after you have clicked on “Reinfer Pattern”.

Inferred Pattern

Inferring the most general pattern. Once the pattern appears, you can specialize the pattern by checking the "Suggest" checkboxes. A checkmark means PatternRefiner should consider including the feature

1

```
// Declaration
try {
    // Guard
    MessageDigest.getInstance(..."SHA-256"...);
    // Post-Method Call
} catch {
    // Exception
} {
    // Error Handling
}
```

Progress:

Match Pattern?	Positive	Unlabelled	Negative
Yes	1	4	1
No	0	16	3

End task

Got stuck? Made a bad choice?  
Restart the task

2

Candidate features for inclusion in the pattern

R: Representativeness  
I: Informativeness

Suggest	Row #	R	I	Feature
<input type="checkbox"/>	0	0.00	0.00	<b>new</b> DigestInputStream(...)
<input type="checkbox"/>	1	0.00	0.00	<b>new</b> DataInfo(...)
<input type="checkbox"/>	2	0.00	0.00	<b>try</b> {
<input type="checkbox"/>	3	1.00	1.00	MessageDigest.getInstance(..."MD5"...)
<input type="checkbox"/>	4	1.00	1.00	MessageDigest.digest()
<input type="checkbox"/>	5	0.00	0.00	MessageDigest.update()
<input type="checkbox"/>	6	1.00	1.00	StringBuilder.toString()
<input type="checkbox"/>	7	0.00	0.00	<b>catch</b> {
<input type="checkbox"/>	8	0.00	0.00	NoSuchAlgorithmException
<input type="checkbox"/>	9	1.00	1.00	IOException
<input type="checkbox"/>	10	0.00	0.00	CertificateEncodingException
<input type="checkbox"/>	11	0.00	0.00	{
<input type="checkbox"/>				<b>new</b> RuntimeException(...)
<input type="checkbox"/>				AbstractEndpoint.getLogCertificate()
<input type="checkbox"/>				<b>new</b> IllegalStateException(...)
<input type="checkbox"/>				}

Optimal  
Redundant  
May be non-optimal

Reinfer pattern

## Candidate features for inclusion in the pattern

R: Representativeness

I: Informativeness

Suggest	Row #	R	I	Feature
<input type="checkbox"/>	0	<div><div></div></div>	0.00	<code>new DigestInputStream(...)</code>
<input type="checkbox"/>	1	<div><div></div></div>	0.00	<code>new DataInfo(...)</code>
<input type="checkbox"/>	2	<div><div></div></div>	0.00	<code>try {</code>
<input type="checkbox"/>	3	<div><div></div></div>	1.00	<code>MessageDigest.getInstance(... "MD5" ...)</code>
<input type="checkbox"/>	4	<div><div></div></div>	1.00	<code>MessageDigest.digest()</code>
<input type="checkbox"/>	5	<div><div></div></div>	0.00	<code>MessageDigest.update()</code>
<input type="checkbox"/>	6	<div><div></div></div>	1.00	<code>StringBuilder.toString()</code>
<input type="checkbox"/>	7	<div><div></div></div>	0.00	<code>} catch (</code>
<input type="checkbox"/>	8	<div><div></div></div>	0.00	<code>NoSuchAlgorithmException</code>
<input type="checkbox"/>	9	<div><div></div></div>	1.00	<code>IOException</code>
<input type="checkbox"/>	10	<div><div></div></div>	0.00	<code>CertificateEncodingException</code>
<input type="checkbox"/>	11	<div><div></div></div>	0.00	<code>) {</code>
<input type="checkbox"/>				<code>new RuntimeException(...)</code>
<input type="checkbox"/>				<code>AbstractEndpoint.getLogCertificate()</code>
<input type="checkbox"/>				<code>new IllegalStateException(...)</code>
<input type="checkbox"/>				<code>}</code>

Optimal

Redundant

May be non-optimal

Reinfer pattern

In the feature feedback pane, the candidate features (A) for inclusion in the pattern are organized as a skeleton, grouped by their relationship (e.g., a guard, exception handling) to the API under analysis.

The features are collected from the entire dataset of examples, and do not necessarily appear in the same examples as one another.

Each feature is displayed with a score indicating their representativeness (R) and informativeness (I):

**Representativeness** measures how common this feature is in the entire population.

**Informativeness** measures how well a feature separates the positive examples from negative examples. Higher is better.

There are two additional analyses: 1) **Impact Analysis** and 2) **What-If Analysis**.

**Impact analysis.** To understand the impact of including a feature, click its name to expand a panel displaying which examples will match, and which examples will no longer match.

☐ 3 0.00 MessageDigest.update()

Match 4 (1 positive, 0 negative, 3 unlabelled) API usages:

Example 1024:  
Positive

```
private static byte[]
computeSha256Digest(byte[] input) {
    try {
        MessageDigest messageDigest =
        MessageDigest.getInstance("SHA-256");
        messageDigest.update(input);
        return messageDigest.digest();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

Example 1002:

```
/**
 * Computes the hash of this URI, with SHA-
 * 256 or MD5 or simple hashCode()
```

but will no longer match 2 (0 positive, 1 negative, 1 unlabelled) API usages:

Example 1013:  
Negative

```
/**
 * Compute the SHA-256 hash of the given
 * byte array
 * @param data the byte array to hash
 * @return the hashed byte array
 * @throws NoSuchAlgorithmException
 */
public static byte[] sha256Hash(byte[]
data) throws NoSuchAlgorithmException {
    MessageDigest messageDigest =
    MessageDigest.getInstance("SHA-256");
    return messageDigest.digest(data);
}
```

Example 1000:

**What if analysis.** When multiple features are selected, some **may** be redundant. Redundant components are colored yellow or red. To inspect the differences between two components, click on the **"Why?"** link that may appear under the suggest checkbox.

☒ 3 0.00 MessageDigest.update()

☒ 4 0.50 MessageDigest.digest()

Why?

This brings up a new panel which shows a side-by-side comparison of the examples matched by one feature, but not the other.

What-If

Comparing examples matching MessageDigest.update() and MessageDigest.digest().

Matched by MessageDigest.update() but not MessageDigest.digest()

Matched by MessageDigest.digest() but not MessageDigest.update()

None

Example 1013:  
Negative

```
/**
 * Compute the SHA-256 hash of the given
 * byte array
 * @param data the byte array to hash
 * @return the hashed byte array
 * @throws NoSuchAlgorithmException
 */
public static byte[] sha256Hash(byte[] data) throws
NoSuchAlgorithmException {
    MessageDigest messageDigest =
    MessageDigest.getInstance("SHA-256");
    return messageDigest.digest(data);
}
```

**Warm up task:** Given some positive and negative examples (examples 1024, 1001, 1009, 1013, 1017; Example 1024 is **positive**, while the rest are **negative**), go to **PatternRefiner** through the provided URL. Once the examples have loaded, **scroll down** to the bottom of the page and **click “Infer pattern”**. Now **wait** as SURF infers a pattern that matches the positive examples.

On the left pane, you can view the pattern inferred by SURF. Initially, this pattern is “`MessageDigest.getInstance(...“SHA-256”)`”. Examples matching this pattern instantiate `MessageDigest` through the `getInstance()` method with “SHA-256” as an argument.

This pattern is a starting point for analyzing the matching examples. The right panel now shows possible features for inclusion in the pattern. To answer questions that require you list examples, identify them by their IDs, e.g. 1024.

1. On the right panel, **click** on the suggestion checkbox for `MessageDigest.update()`, and `MessageDigest.digest()`. (typically, they are found between **Rows 4--6**)  
After **PatternRefiner** has performed its computations in the background, click on “Why?” under the checkbox in **Row 6** to bring up the What-If Analysis. **Which negative examples** call `MessageDigest.digest()` but do not call `MessageDigest.update()`?
2. **Close** the What-If Analysis from Question 1. **Uncheck** the rows from Step 1.

**Click** on the text “`new RuntimeException(...)`” to expand the Impact Analysis panel, which displays the examples matched and not matched by the feature. **Which positive example constructs** a `RuntimeException`? (i.e., has “`new RuntimeException(...)`”)

3. **Close** the Impact Analysis Panel from Question 2 by clicking on the text “`new RuntimeException(...)`” in **Row 11**.  
You can see the exceptions handled among the examples matching the pattern (usually in **Rows 8-10**, under the “`catch (`”). Judging by the **informativeness** of the exception-handling features, **can at least one pair of positive and negative examples** be distinguished from one another if the pattern included `NoSuchAlgorithmException` (i.e., is its informativeness greater than 0)?
4. **Click** on the text `NoSuchAlgorithmException`, in **Row 8**. This expands the impact analysis panel, which displays the examples matched and not matched. **Which unlabelled examples match** `NoSuchAlgorithmException`?

Finally, to **reinfer** the pattern with your feedback on which feature to include,

5. **Close** the Impact Analysis Panel from Question 4.  
Given that we wish to focus on examples that catch the **same exceptions** as the given positive examples, **click** the suggest checkbox of the exception with the **highest informativeness**, and **click** on “**Reinfer Pattern**”. After the pattern has been updated, **click** on “**End Task**” (on the left panel).